PRIVACY-PRESERVING COLLABORATIVE ANALYTICS OF LOCATION DATA

A DISSERTATION SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

By
Emre Yılmaz
September 2017

PRIVACY-PRESERVING COLLABORATIVE ANALYTICS OF LOCATION DATA

By Emre Yılmaz

September 2017

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Erman Ayday(Advisor)

Hakan Ferhatosmanoğlu(Co-advisor)

Özgür Ulusoy

Engin Demir

Ali Aydın Selçuk

İbrahim Körpeoğlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

PRIVACY-PRESERVING COLLABORATIVE ANALYTICS OF LOCATION DATA

Emre Yılmaz
Ph.D. in Computer Engineering
Advisor: Erman Ayday
Co-advisor: Hakan Ferhatosmanoğlu
September 2017

Deriving meaningful insights from location data helps businesses make better decisions. While businesses must know the locations of their customers to perform location analytics, most businesses do not have this valuable data. Location data is typically collected by other services such as mobile telecommunication operators and location-based service providers. We develop scalable privacy-preserving solutions for collaborative analytics of location data. We propose two classes of approaches for location analytics when businesses do not have the location data of the customers. We illustrate both of our approaches in the context of optimal location selection for the new branches of businesses. The first type of approach is retrieving the aggregate information about the customer locations from location data owners via privacy-preserving queries. We define aggregate queries that can be used in optimal location selection and we propose secure two-party protocols for processing these queries. The proposed protocols utilize partially homomorphic encryption as a building block and satisfy differential privacy. Our second approach is to generate synthetic location data in order to perform analytics without violating privacy of individuals. We propose a neighborhood-based data generation method which can be used by businesses for predicting the optimal location when they have partial information about customer locations. We also propose grid-based and clustering-based data generation methods which can be used by location data owners for publishing privacy-preserving synthetic location data. Proposed approaches facilitate running optimal location queries by businesses without knowing their customers' locations.

Keywords: Data Privacy, Location Analytics, Optimal Location Queries, Differential Privacy, Homomorphic Encryption, Data Generation, Uncertainty.

ÖZET

KONUM VERİSİNİN GİZLİLİĞİNİN KORUNARAK ORTAKLAŞA ANALİZİ

Emre Yılmaz
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: Erman Ayday
İkinci Tez Danışmanı: Hakan Ferhatosmanoğlu
Eylül 2017

Konum verilerinden anlamlı çıkarımlar yapmak işletmelerin daha iyi kararlar vermelerine yardımcı olmaktadır. Konum analizi yapabilmek için işletmelerin müşterilerinin konumlarını bilmeleri gerekmesine rağmen çoğunlukla işletmeler bu değerli veriye sahip değillerdir. Konum verileri genellikle mobil telekomünikasyon operatörleri ve konum tabanlı servis sağlayıcılar tarafından toplanmaktadır. Bu tezde, konum verisinin ortaklaşa analiz edilebilmesi için ölçeklenebilir ve gizliliği koruyan çözümler geliştirilmiştir. İşletmelerin müşterilerinin konum bilgilerine sahip olmadıklarında kullanabilecekleri iki farklı yaklaşım türü önerilmektedir. Önerilen yaklaşımlar şirketlerin yeni şubeleri için en iyi yeri bulması problemi bağlamında açıklanmaktadır. İlk yaklaşım türü, gizliliği koruyan sorgular aracılığıyla konum verisi sahibinden müşteri konumları hakkında toplu bilgiler elde etmektir. Bu amaçla en iyi yer seçiminde kullanılabilecek toplu sorgular tanımlanmış ve bu sorguları cevaplayabilmek için güvenli iki taraflı protokoller geliştirilmiştir. Önerilen protokoller kısmi homomorfik şifreleme kullanılarak geliştirilmiştir ve ayrımsal gizliliği sağlamaktadır. İkinci yaklaşım ise bireylerin gizliliğini ihlal etmeden analiz yapmak için sentetik konum verisi yaratılmasıdır. İşletmelerin müşterilerinin konumları hakkında kısmi bilgiye sahip olduklarında en iyi yeri tahmin etmek için kullanabilecekleri komşuluk tabanlı veri üretimi yöntemi önerilmiştir. Ayrıca, konum verisi sahiplerinin, gizliliği koruyan sentetik konum verisi paylaşımında kullanabilecekleri karelere bölme ve kümeleme tabanlı veri üretim yöntemleri de önerilmiştir. Önerilen yaklaşımlar işletmelerin müşterilerinin konumlarını bilmeden en iyi yer seçimi yapmalarına yardımcı olacaktır.

Anahtar sözcükler: Veri Gizliliği, Konum Analizi, En İyi Yer Sorguları, Ayrımsal Gizlilik, Homomorfik Şifreleme, Veri Üretimi, Belirsizlik.

Acknowledgement

I would like to thank to my advisors, Hakan Ferhatosmanoğlu, Ali Aydın Selçuk, and Erman Ayday, for guiding and supporting me during my PhD studies. It has been a great pleasure to work with them. I would like to express my deepest gratitude to Hakan Ferhatosmanoğlu for his patience, guidance, and encouragement.

I would also like to thank my thesis monitoring committee members Özgür Ulusoy and Engin Demir for their valuable comments and discussion. I would like to thank my jury member İbrahim Körpeoğlu for his remarks and suggestions.

I am thankful to Remzi Can Aksoy and Sanem Elbaşı for their support and contribution to this work.

I want to thank Türk Telekom for their financial support in part.

Finally, I would like to thank my lovely family for their love, endless support, and encouragement. In particular, I would like express my great appreciation and dedicate this thesis to my beloved wife Ayşe and my dear daughter Betül İpek.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Location analytics is the process or the ability to gain insight from the location data. Businesses use location analytics in many ways [1] such as finding the best place to locate a new facility, identifying the performances of stores, analyzing sales in different regions to offer products and prices most suitable for these regions, and managing insurance risks based on the potential of disasters in given locations. Many vendors such as Alteryx, Esri, and Pitney Bowes provide location analytics applications. Businesses can use these applications to analyze the locations of their customers when the location datasets are available. However, customer locations are not known by businesses most of the time. In this dissertation, we address the problems of performing a variety of location analytics tasks when the customer locations are not available in house. Particularly, we consider the problem of selecting the optimal location which is a common location-based analysis that seeks the best location to open a new facility optimizing an objective function given a set of existing facilities and a set of customers.

Previous works on optimal location queries in the data management community focus on returning the best candidate as fast as possible [2, 3, 4]. Some of

these works select the optimal location from a given region, whereas the others select from a set of candidate locations. The common approach is to use pruning based algorithms and index structures to decrease the processing times, instead of sequentially checking each possible location. The methods in the literature mostly find the optimal location when the locations of existing facilities and the locations of customers are given. Hence, businesses need to know the locations of their customers in order to use these algorithms. However, this is rarely the case. Most businesses do not have the knowledge of customer locations. For example, fast food restaurant chains or coffeehouse chains typically do not know the addresses of their customers. Therefore, when these businesses plan to open new branches, they cannot directly use the existing techniques for finding the optimal location.

Location data is typically collected by mobile telecommunication operators and service providers, such as Foursquare. The data owners also seek ways to enable other businesses to utilize their location data without violating their customers' privacy. One needs to prevent the location-based service providers from tracking the users individually, while still allowing other businesses to obtain useful information. Similarly, businesses do not want to share their customer lists with location-based service providers. In this thesis, we develop efficient privacy-preserving solutions that help to identify the best locations to open new branches when the customer locations are not known by businesses.

1.2 Problem Definition

In this dissertation, we refer the location data owner as the server, and the business that wants to do location analytics as the client. We refer their customers as the users of the server and the users of the client. The client has existing facilities, such as branches of a bank, and aims to find the optimal location for the new one among several candidates. The client does not know the locations of its users and location data is stored in the server. The client may know some partial information about locations of the users and it can be used in optimal

location selection. The problem is then to find the optimal location for a new facility without knowing the locations of the users.

Our first approach for doing location analytics without location data is executing privacy-preserving queries over the server. The server cannot share its location data with the client due to privacy and legal concerns. Hence, the main privacy requirement is hiding location data from the client. We define a fundamental class of queries that can be used in optimal location selection. In these queries, the client only obtains aggregate information about locations of its users without learning the location of any specific user. The client has several candidates for the new facility and it can request the queries for each candidate location and select the best one. Other than the location data, the client's user list and the server's user list must be hidden from each other. In addition, the result of the query must be hidden from the server.

We propose server-based and client-based query processing protocols utilizing homomorphic encryption as building block. The main difference of these two types of protocols are the cryptographic keys used in the protocols. In server-based protocols, most of the computation is performed by the server since its cryptographic keys are used in the protocols. Client-based protocols requires less communication and computation during query processing in which most of the computation is performed by the client in the setup phase. We formally prove that the privacy-preserving protocols hide the sensitive data from unauthorized parties and do not leak any information other than the output of the protocol. To prevent information leaks from the query results, we also satisfy differential privacy in the proposed protocols.

Our second approach is using synthetic data in location analytics when the original data is not available. Synthetic data can be generated either by the client or the server, depending on the requirements. We first focus on the client side and propose a neighborhood-based data generation method. This can be used when the client has partial information about user locations such as the density of users in existing facilities. Such partial information can also be obtained by running proposed query processing protocols. Inside the Voronoi region of each

facility, the proposed data generator creates more customers in the subregions with boundary to higher density neighbors by dividing each Voronoi region into triangular subregions. We define the auxiliary information that may be known by businesses. The data generator uses provided auxiliary information and generates user locations. After data generation, the query processor runs an optimal location query with generated data and returns the best candidate.

Since the server owns real location data, it can anonymize data and publish it. Even though it is possible to do location analytics on anonymized data, it can be vulnerable to de-anonymization attacks [5]. Hence, we also propose generating synthetic location data based on the characteristics of original location-based information at the server side. The synthetic data generated by the server need to provide good privacy preservation with high utility. We describe two types of methods based on grid-based data generation and clustering-based data generation. Both of these approaches satisfy differential privacy and provides high utility in optimal location queries.

1.3 Contributions

The key contributions of the dissertation are summarized as follows:

- We enhance facility location problems by removing the assumption that the customer locations are known to businesses. With the proposed solutions, a business can find the best location for a new facility among several candidates without knowing its customer locations.
- In Chapter 3, we introduce query processing protocols for different types of queries, i.e., RNN cardinality query, average distance query, and maximum distance query that can be used as a service to identify optimal facility location. Our protocols utilize homomorphic encryption for protecting privacy of both parties and satisfy differential privacy.

- The proposed query processing protocols take advantage of using a potential superset of user space to hide the user lists of both parties. Our solution does not use any computationally expensive cryptographic comparisons such as private equality testing or private set intersection. The performance evaluations show that the proposed protocols are practical, efficient, and scalable. For instance, when the server has 25 million users, executing privacy-preserving RNN cardinality query takes around 10 seconds on a modest computer.

- In Chapter 4, we develop an optimal location predictor for choosing a location for the new facility by generating customer locations based on the density of the customers in each existing facility and the given auxiliary information. Our experiments with real location data from New York City and Tokyo show that the proposed predictor finds the optimal location for the new facility among several candidates even though the customer locations are not known.

- In Chapter 4, we also propose privacy-preserving synthetic data generation methods for location data which eliminate the risks of de-anonymization while allowing businesses to analyze user locations. Generated data protects the privacy of the individuals in the database by satisfying differential privacy and data owners can publish the generated data to other parties without privacy concerns.

Parts of this dissertation were published in [6] and [7].

1.4 Outline

The rest of the dissertation is organized as follows. A literature review and background information are given in Chapter 2. Aggregate queries for optimal location selection are defined and privacy-preserving query processing protocols are presented in Chapter 3. In Chapter 4, we present the optimal location predictor

which accepts partial information about user locations and returns a location for the new facility. We also explain privacy-preserving synthetic location data generation methods in Chapter 4. Finally, we conclude the dissertation in Chapter 5.

Chapter 2

Related Work and Background

In this chapter, we first give the literature review of optimal location queries in Section 2.1. Since we first aim to run privacy-preserving queries on the server to find the optimal location, we summarize the existing work on privacy-preserving location-based query processing in Section 2.2. Then, the concept of differential privacy and homomorphic encryption schemes are explained in Section 2.3 and Section 2.4, respectively, as building blocks of our query processing protocols. We review the query processing methods on uncertain databases in Section 2.5. Finally, privacy-preserving data generation methods in the literature are explained in Section 2.6.

2.1 Optimal Location Queries

Nearest neighbor (NN) query is a well-studied problem with many variants in the literature [8, 9, 10]. Reverse nearest neighbor (RNN) query finds the set of points that have the query point as the nearest neighbor [11]. In most of the real-life applications bichromatic reverse nearest neighbor (BRNN) query is used. In BRNN, points are divided into two categories such as users and facilities. Given a facility f, BRNN query finds the set of users that have f as the nearest

facility. BRNN query is a fundamental query for optimal location studies because generally it is assumed that each user prefers her closest facility. Hence, BRNN of a facility is the set of users who are attracted by that facility.

Identifying the optimal location for a new facility has been widely studied in the literature with applications in decision-support systems and strategic planning of businesses. An optimal location query asks for a location to build a new facility that optimizes an objective function. For different types of facilities, different objective functions were defined in the literature. The mostly studied objectives are (i) *max-inf:* maximizing total number of users attracted by the new facility and (ii) *min-dist:* minimizing the average distance between each user and her nearest facility.

Max-inf optimal location query: Given a set $\mathcal{F}$ of existing facilities and a set $\mathcal{U}$ of users, max-inf optimal location query finds a location p for the new facility with maximum influence. In [2], the influence of a location is defined as the total weight of its BRNN. Each user has a weight and the query computes a location p in a given region Q which maximizes the total weight of users who are closer to p than to any facility. The problem is studied in L_1-norm space and the authors propose methods using different index structures such as R^*-tree, OL-tree and virtual OL-tree. Maximizing the BRNN of the new facility in L_2-norm space is studied in [12]. Utilizing the region-to-point transformation, the authors solve the problem by searching a limited number of points instead of searching all possible points in the space. The same problem is studied assuming that each facility has a given capacity [13]. Another study returns top-k locations from a set of candidate locations instead of the best one [14]. The general assumption in optimal location queries is that each user prefers her closest facility. In [15], it is assumed that a user tends to go to her k nearest facilities. Hence, a facility attracts users if the facility is one of her k nearest facilities. They find an optimal location such that setting up a new facility attracts the maximum number of users.

Min-dist optimal location query: Given a set $\mathcal{F}$ of existing facilities and a set $\mathcal{U}$ of users, min-dist optimal location query finds a location p such that the

average distance from each user to her closest facility is minimized if the new facility is built at location p. This query is widely used in real-life applications to improve the quality of service or reduce the logistics cost by businesses. It is firstly defined in [3] to select the min-dist optimal location from a given region. Although there are infinite number of locations in a region, the authors prove that it is possible to limit the number of candidate locations in L_1-norm space and the exact result is included in finite number of candidate locations. Qi et al. [4] solve the problem in L_2-norm space for the set of candidate locations and investigate the variant of the problem called min-dist facility replacement problem. Instead of adding a new facility, replacing a facility is aimed in facility replacement problem. Algorithms to solve optimal location queries in road networks have also been studied [16].

Two other objectives in optimal location queries are minimizing the maximum distance between a user and her closest facility [16, 17] and uniformly distributing users into facilities. Previous works on optimal location queries select the optimal location either from a given region [3, 12] or from a candidate location set [4, 14]. When it is selected from a given region, infinite number of candidate locations is firstly limited. Then it becomes possible to search limited number of candidates. In this thesis, we select the optimal location from given candidate locations because businesses typically choose the facility locations from several candidates in practice. In addition, existing works focus on efficiently returning the best candidate using pruning techniques and index structures. However, they return the optimal location when the exact customer locations are given. Our work differs from existing works because we remove the the assumption that the customer locations are known to businesses. We introduce a new problem setting in which businesses may only know partial information about customer locations and the exact locations are stored in location data owners.

2.2 Location Privacy

Today, vast amounts of information are collected and analyzed in databases around the world. Data may be stored by multiple parties and these parties may not be keen on sharing their data with others. In secure multi-party computation (SMC), multiple parties jointly compute a function over their inputs without revealing their inputs to each other. In [18], several SMC problems are identified. One such problem defined in [18] is the privacy-preserving database query, where Alice seeks a match with her private string q in Bob's database T. The privacy requirement is hiding q and the query result from Bob, and hiding T from Alice. The authors develop an efficient solution for the matching problem in [19] by using a semi-trusted third party.

Privacy-preserving location-based queries have been studied in the literature. Cheng et al. [20] propose a privacy-preserving range query protocol to find users within a range with non-zero probability. In [20], each user has a cloaked region to hide her exact location, and the probability of being within a range depends on the intersection of the cloaked regions. A hybrid approach that integrates private set intersection and location cloaking is presented in [21]. For privacy-preserving NN queries, a privacy-aware query processing framework called Casper is presented in [22]. This framework uses a location anonymizer to blur users' exact locations into cloaked regions. Ghinita et al. [23] eliminate the usage of third-party anonymizers by using cryptographic techniques. They utilize private information retrieval techniques to preserve location privacy. In [24], efficient protocols are proposed for privacy-preserving k-NN searches by using several primitive SMC protocols. Yi et al. [25] present solutions for the same problem and use Paillier encryption and location cloaking as building blocks.

Existing works on location privacy try to hide the location information that the client (i.e. querying side) has from the server (i.e. location-based service provider). In our scenario, user location information is stored in the server and the server hides this sensitive information from the client. The client wants to

analyze user locations in order to find the optimal facility location. In Chapter 3, we propose secure two-party protocols that allow analyzing location data in the server. In Chapter 4, we propose privacy-preserving synthetic location data generation methods that aim to release data without violating privacy of individuals.

2.3 Differential Privacy

Differential privacy aims to protect the privacy of individuals while releasing aggregate information about the database. It is based on the neighborhood of databases. Two databases D and D' are neighbors if they differ in only one entry. Differential privacy requires that query results for two neighbor databases should be indistinguishable. Let the output of a protocol P on database D be $P(\mathtt{D})$. The differential privacy is formally defined as follows:

Definition 1 *Protocol P satisfies ϵ-differential privacy if for any two neighbor databases D and D', and any subset S of output space of P,*

$$\Pr\left[P(\mathtt{D}) \in S\right] \leq \Pr\left[P(\mathtt{D'}) \in S\right] \cdot e^{\epsilon}$$

A typical way to achieve differential privacy is adding controlled random noise to the query result. For numeric queries, Laplace mechanism can be used to produce the noise drawn from the Laplace distribution. Let $Laplace(\lambda)$ be a sample from Laplace distribution with mean 0 and standard deviation λ. To obtain ϵ-differential privacy, the noise drawn from the Laplace distribution must be calibrated according to the sensitivity of the protocol [26]. The sensitivity of the protocol is the maximum possible change on the output by changing a single record in database. Given a protocol P, the sensitivity of the protocol is defined as follows:

Definition 2 *Let $\mathcal{N}$ be the set of all pairs of neighbor databases.*

$$\Delta P = \max_{(\mathtt{D},\mathtt{D'})\in\mathcal{N}} \|P(\mathtt{D}) - P(\mathtt{D'})\|$$

Therefore, a protocol P satisfies ϵ-differential privacy for the result

$$P(\mathtt{D}) + Laplace(\frac{\Delta P}{\epsilon})$$

In Chapter 3, we define aggregate queries that can be used for optimal location selection and we explain how to satisfy differential privacy in these protocols. We also explain differential privacy in the context of synthetic data release in Chapter 4.

2.4 Homomorphic Encryption

In homomorphic encryption, a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext. Cryptosystems that allow homomorphic computation for a limited number of operations such as addition or multiplication are called partially homomorphic. For instance, given two messages x and y, one can compute the encryption of $x + y$ by using the encryptions of x and y in an additive homomorphic encryption scheme. In multiplicative homomorphic schemes, $E(x \cdot y)$[1] can be computed by using $E(x)$ and $E(y)$. Gentry [27] proposed first fully homomorphic encryption scheme that supports both addition and multiplication. Since partially homomorphic schemes are more efficient and calculating the sum is sufficient for our query processing protocols in Chapter 3, we are interested in additive homomorphic cryptosystems [28, 29, 30], satisfying $E(x) \cdot E(y) = E(x + y)$. Another homomorphic property of these cryptosystems is that encrypted plaintext $E(x)$ raised to a constant k is equal to encryption of the product of the plaintext x and the constant k, i.e. $E(x)^k = E(x \cdot k)$.

We develop our query processing protocols by using the Paillier cryptosystem [30]. In Paillier, if the public key (PK) is the modulus m and the base g, then the encryption of a message x is $E(x) = g^x \cdot r^m \pmod{m^2}$, for some random $r \in \{0, ..., m-1\}$. Using a random value r in encryption ensures that two messages

[1]For the rest of the thesis, $E(x)$ denotes the encryption of message x

that are the same will encrypt to the same value with only a negligible likelihood. Hence, Paillier provides *semantic security*. m should be selected as the product of two primes p and q. The private keys (SK) of the Paillier cryptosystem are $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^{\lambda} \bmod m^2))^{-1} \bmod m$, where $lcm(a, b)$ is the least common multiple of a and b, and $L(u) = \frac{u-1}{m}$. The decryption of a ciphertext c can be performed using private keys as follows: $D(c) = (L(c^{\lambda} \bmod m^2) \cdot \mu) \bmod m$. Paillier satisfies $E(x) \cdot E(y) = E(x+y)$, because $(g^x \cdot r_1^m) \cdot (g^y \cdot r_2^m) = g^{x+y} \cdot (r_1 + r_2)^m$. As a result of this homomorphic property, multiplying a ciphertext $E(x)$ with $E(0)$ creates another ciphertext which is the fresh encryption of x.

2.5 Query Processing Over Uncertain Data

Query processing over uncertain data has been studied in the literature for different type of queries. Wang et al. [31] presents a survey about data uncertainty and the types of uncertain data queries. Uncertain top-k query returns most probable top-k answers [32]. Soliman et al. [32] propose query processing algorithms in which the answer of the query depends on both the tuple scores and probabilities. Tao et al. [33] define range queries on uncertain databases to return objects in a given region whose probability is greater than a given threshold, where each object has an imprecise location. They propose the concept of probabilistically constrained rectangle and an index structure U-Tree for efficiently processing uncertain range queries. The probabilistic nearest neighbor query is firstly proposed in [34]. In order to return all objects which can be the nearest neighbor of the query point with non-zero probability, their algorithm performs a pruning of objects which do not have a chance of nearest neighbor of the query point. Cheema et al. [35] formalize probabilistic reverse nearest neighbor query that returns the objects which can be the RNN of the query point with higher probability than a given threshold. They propose an algorithm using several pruning techniques such as half-space pruning, dominance pruning, metric-based pruning, and probabilistic pruning. Li et al. [36] investigate the problem of probabilistic RkNN query and proposes an efficient and scalable algorithm using probabilistic

pruning and spatial pruning techniques. In all of these works, objects are associated with probabilities and the query results are computed based on these probabilities. Their approaches cannot be directly applied to our problem when the client has partial information about users because there is no probability associated with user locations. The only known information is the number of users attracted by each existing facility. Hence, a user can be located at any point in the Voronoi region of her nearest facility. To the best of our knowledge, our work of predicting the optimal location using partial information, which is presented in Chapter 4, is the first to address processing of optimal location queries under such uncertainty.

2.6 Privacy-Preserving Synthetic Data Generation

With the advances in information technologies, organizations collect, store, and process large amount of data. Data owners cannot share their data with other parties since sensitive and private information are contained most of the time. Organizations may want to share their data with third parties for research and innovation purposes. In the literature, several approaches have been proposed for privacy-preserving data publication such as data anonymization [37, 38, 39] and data perturbation [40]. However these approaches have a risk of de-anonymization [5] or recovery [41]. Synthetic data generation is a safer alternative against these approaches. In synthetic data generation, data is randomly generated based on a model that keeps some statistical information from the original data. The synthetic data can allow analytics by avoiding inclusion of sensitive private information.

In [42], model based data generation is proposed as an alternative approach to data perturbation. Vreeken et al. propose a Minimum Description Length (MDL) based algorithm to generate privacy-preserving synthetic data. Using the frequency of each item set, their algorithm assigns a probability and generates

data randomly. The proposed approach is not appropriate for location data. Some other methods for specific data types are proposed [43, 44], which are not applicable to location data.

For spatial data, Xiao et al. [45] focus on developing summaries that contain noisy counts in multi-dimensional regions. They propose to publish differentially private noisy counts of data points in a multi-dimensional partitioning structure [45]. Cormode et al. [46] develop differentially private spatial decompositions, such as quadtrees and kd-trees, by setting non-uniform noise parameters in a hierarchical structure.

Lu et al. [47] define the model of the database by a set of counting queries such as the number of male customers and the number of orders from male customers. They add noise to the result of the defined queries to satisfy differential privacy and then generate a synthetic database based on noisy query results. This data generation technique can be used with the differentially private partitioning approaches in [45] and [46] to answer the defined counting queries. However, the synthetic data is specifically generated for the defined queries and it can be efficiently used to answer the defined queries. In Chapter 4, we propose methods for generating generic differentially private synthetic location data that allows a wide variety of analytics tasks besides counting queries.

Chapter 3

Privacy-Preserving Query Processing for Optimal Location Selection

In this chapter, we define a fundamental class of queries that can be used in optimal location selection [6]. In these queries, the client only obtains aggregate information about locations of its users without learning the location of any specific user. A simple example to these aggregate queries is average distance query, in which the client retrieves the average distance of its users to their nearest facilities. The nearest facility of each user is the facility that has the minimum distance to that user. The average distance is a valuable information for the client to minimize it for maximizing user benefit. In a non-privacy-preserving solution for this query, the client sends the facility locations and its user list to the server. The server checks the location of each user (who gave informed and explicit consent for this information) and calculates distances to their nearest facilities. At the end of the query, the server returns the average distance and the client obtains useful information for facility location without tracking its users individually. The client can send a different location for the new facility in each query together with the locations of existing facilities. As a result, it can select the best candidate that minimizes the average distance between users and their

nearest facilities.

For a privacy-preserving solution, we need to hide the client's user list and the server's user list from each other. We also need to hide the answer to the query from the server. Otherwise, the server learns the best candidate for the new facility and it may share this information with the competitors. We investigate privacy-preserving solutions to aggregate queries which allow analyzing location data in a server and selecting the best facility location. With the proposed solutions, without sharing its user list with the server, the client can obtain aggregate information about user locations and find an optimal place for its new facility among several candidates for different objective functions. These objectives are (i) uniformly distributing the cardinality of the reverse nearest neighbors (RNN), (ii) minimizing the average distance between each user and her closest facility, and (iii) minimizing the maximum distance between a user and her closest facility.

We define three fundamental aggregate queries for optimal location selection and propose two types of privacy-preserving query-processing protocols for each type of query, utilizing partially homomorphic encryption as a building block. We encrypt the sensitive data of the server and the client, and perform the operations on the encrypted data to preserve the privacy of both parties. First, we explain server-based protocols, in which most computation is performed by the server, and hence the workload of the client is low. This solution is particularly convenient when the client has limited computational power. To decrease the communication overhead in each query, we also propose client-based protocols. In these protocols, the client performs the majority of the computation during the setup phase (which occurs only once). After completion of the setup phase, all queries are processed with low communication overhead. Therefore, our client-based solution is highly efficient when the client undertakes some pre-computations before running its queries.

During the protocols, homomorphic encryption is used for keeping the user list of the client and the query result hidden from the server and keeping the user list of the server and location data hidden from the client. Initially, we describe the

protocols to return exact query results. Since the server is unaware of the query result and the queries return aggregate results, some queries may leak information about users. For instance, if the result of a counting query is one, that user can be predicted by the client. To prevent information leak about any single user, we also satisfy differential privacy in our protocols by adding controlled noise to the query result. Therefore, we use homomorphic encryption and differential privacy together to guarantee privacy of individuals during query processing.

3.1 Problem Formulation

We present our system model in Section 3.1.1. Formal definitions of the queries are given in Section 3.1.2. We describe the threat model in Section 3.1.3.

3.1.1 System Model

There is a server ($\mathcal{S}$) (e.g., a location-based service provider) that provides analytics as a service and a client ($\mathcal{C}$) that requests queries. The server is the database owner and has n_s users $\mathcal{U}_\mathcal{S} = \{S_1, S_2, ..., S_{n_s}\}$. In addition, the server has location information for each S_i at different time periods. The client has n_c users $\mathcal{U}_\mathcal{C} = \{C_1, C_2, ..., C_{n_c}\}$ and a list of its k existing facilities $\mathcal{F} = \{F_1, F_2, ..., F_k\}$. The locations of the existing facilities are public and known by the server. The client wants to run aggregate queries such as count, sum, and maximum on the location data of the server, e.g., to analyze the candidate locations for a new branch. The client aims to hide $\mathcal{U}_\mathcal{C}$ and the query results from the server. The server also aims to hide $\mathcal{U}_\mathcal{S}$ from the client and prevent user tracking by the client. Hence, the client will not learn anything about the location of any specific user; it will only obtain the query result at the end of the protocol.

We sketch out our system model in Figure 3.1. To run aggregate queries about its users, the client must identify its users in $\mathcal{U}_\mathcal{S}$ using an identifier. Before running queries, the server and the client decide on an identifier such as mobile phone

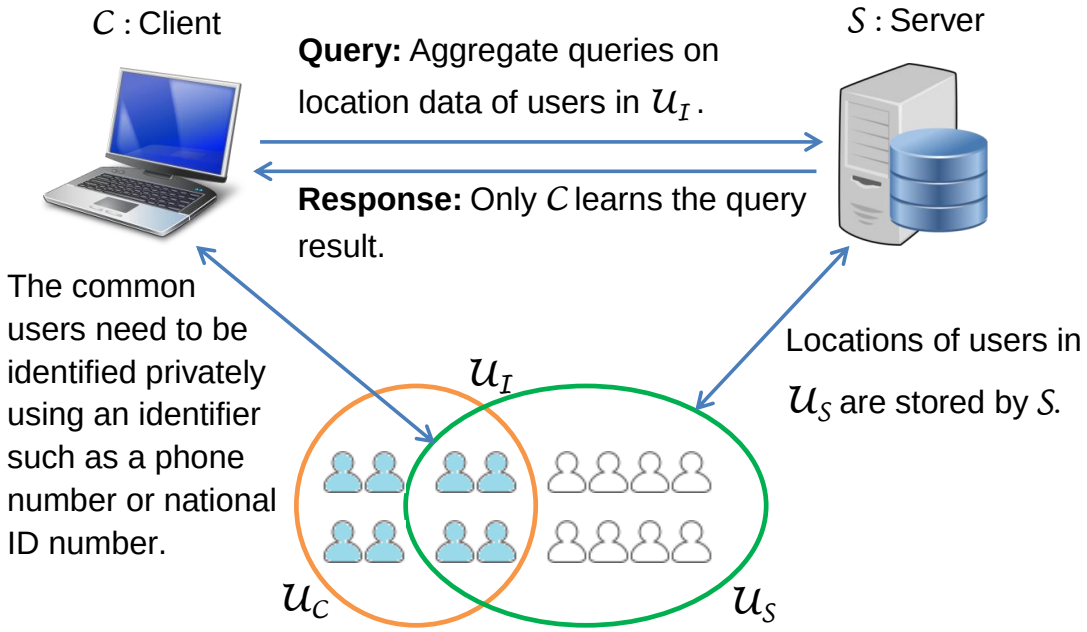


Figure 3.1: System model for privacy-preserving query processing protocols.

number. Most businesses and service providers know mobile phone numbers of their customers. Another identifier can be national identification number. If the server is a telecommunication company and the client is a bank or a hospital they might use national identification number as the identifier. Let $\mathcal{U}_\mathcal{I}$ be $\mathcal{U}_\mathcal{S} \cap \mathcal{U}_\mathcal{C}$, and n_I be the cardinality of $\mathcal{U}_\mathcal{I}$. Since the server does not have the location information of users in $\mathcal{U}_\mathcal{C} \backslash \mathcal{U}_\mathcal{S}$, we define our queries for the users in $\mathcal{U}_\mathcal{I}$.

We define three useful types of queries for this context: RNN Cardinality Query (RNNQ), Average Distance Query (AVGQ), and Maximum Distance Query (MAXQ). Since the server knows the user locations, it can calculate the distance between a user and a facility via any distance measure. The main challenges are keeping $\mathcal{U}_\mathcal{C}$ hidden from the server and preventing user tracking by the client.

We propose two types of solutions for each query type, the server-based solutions and the client-based solutions. The server is responsible for most of the computation in the server-based solutions. Hence, they are suitable when the client prefers outsourcing computation. The drawback of server-based solutions

over the client-based version is their communication overhead. The client-based solutions reduce communication overhead significantly. In the client-based solutions, most of the computation is performed by the client only in the setup phase. In Section 3.2 and Section 3.3, we describe the server-based and the client-based protocols which return exact query results. Since exact query results may leak information in some cases such as counting queries, in Section 3.4 we explain how to add controlled random noise to the query result in each protocol to satisfy differential privacy.

3.1.2 Query Definitions

3.1.2.1 RNN Cardinality Query (RNNQ)

One of the objectives of optimal location queries is uniformly distributing the workload in facilities. In this case, the new facility should attract users from dense facilities. Attracting a user is equivalent to being the closest facility to the user. This query finds the number of users attracted by each facility. The formal definition of the RNNQ is as follows:

Query 1 *Given facility locations, find the total number of users in $\mathcal{U}_\mathcal{I}$ attracted by each facility. In other words, calculate the cardinality of RNN for each facility.*

In practice, the client can initially run the RNNQ with existing facilities $\mathcal{F}$ to analyze the distribution of the users. Using the result, the client can determine candidate locations for the new facility F_{k+1}. For candidate locations, the client can run the RNNQ with $\mathcal{F} \cup F_{k+1}$. Hence, the client can observe the total number of users attracted by each candidate location for F_{k+1} and select the location that provides the most balanced distribution.

3.1.2.2 Average Distance Query (AVGQ)

One of the objectives of optimal location queries is minimizing the average distance between each user and her closest facility. For instance, delivery services pay attention to decreasing the average distance between their customers and the nearest shop. The AVGQ is formalized as follows:

Query 2 *Given facility locations, find the average distance between users in* $\mathcal{U}_{\mathcal{I}}$ *and each one's nearest facility.*

In practice, the client can run the AVGQ with $\mathcal{F} \cup F_{k+1}$, where F_{k+1} is a candidate location for the new facility. Hence, the client can select the optimal location for F_{k+1}, which minimizes the average distance.

3.1.2.3 Maximum Distance Query (MAXQ)

Another objective of optimal location queries is minimizing the maximum distance between a user and her closest facility. In this objective, the aim is to optimize the worst-case cost of reaching the nearest facility. The MAXQ is formalized as follows:

Query 3 *Given facility locations, find the maximum distance between a user in* $\mathcal{U}_{\mathcal{I}}$ *and her nearest facility.*

In practice, the client can run MAXQ with $\mathcal{F} \cup F_{k+1}$, for candidate F_{k+1} locations. The client can select the optimal location for F_{k+1}, which minimizes the maximum distance.

3.1.3 Threat Model

In our model, both the server and the client are considered "semi-honest". Therefore, both parties follow the protocol correctly; however, they may try to learn additional information by analyzing the data. That is, the server may try to determine the client's user list, and similarly, the client may try to determine the individual locations of its users during the protocol (by using the messages they receive throughout the protocol). On the other hand, both the server and the client follow protocol execution honestly by forming correct messages, input, and output parameters for each other. This is a reasonable assumption in the problem setting since both parties are motivated to produce the correct result. The server sells the service and the correct result increases the client's satisfaction. Also, the client finds the best facility location if the query results are correctly calculated.

The proposed solutions are secure two-party protocols in which the server and the client wish to compute the query result securely without sharing their inputs with the opposing party. Both the server and the client have sensitive data that should be hidden from the other party. We formally list the sensitive data as follows:

1. Input of the client: $\mathcal{U}_{\mathcal{C}}$.
2. Input of the server: (a) $\mathcal{U}_{\mathcal{S}}$ and (b) **location information of users in** $\mathcal{U}_{\mathcal{S}}$.
3. Output of the protocol: **Query result.**

We aim to hide all of the above sensitive data (from unauthorized parties) in our protocols. The parties must not learn the input of each other. At the end of the protocols, only the client must get the query result and the server must not learn it. The privacy of the server is assured if sensitive data 2 is hidden from the client, and the privacy of the client is assured if sensitive data 1 & 3 are hidden from the server. We prove the security of our proposed protocols in the semi-honest model using the simulation paradigm defined in [48].

While the locations of existing facilities are typically public, the location of a new facility can be sensitive data for the client. In this case, the client can run the query with some dummy locations to provide K-anonymity [49], which provides indistinguishability among K locations. Since the query result is hidden from the server, all of the K locations are indistinguishable for the server.

One potential threat to the server's sensitive data may be obtaining information via exhaustive client queries. By using non-existing facilities, the client can try to obtain information about location of some users. For instance, the client can divide the whole region into two regions and select the center of each region as a facility location. When the client performs RNNQ with these facility locations, it learns the total number of users in each region. The client can divide each region into smaller regions in subsequent queries, until each region has at most one user. At the end, the client learns the small regions which contains a user and it may predict the user in a small region with background knowledge. Therefore, if the total number of facilities in the query is very small or very large, the client may obtain information about user locations.

We assume the locations of k existing facilities of the client are public and known by the server. The server decides two threshold values θ_1 and θ_2 such that the client can add at most θ_1 new facilities or remove at most θ_2 existing facilities in a query[1]. Thus, when the client sends the locations of the facilities, the server aborts the protocols in following cases:

- if the total number of facilities is greater than $k + \theta_1$,
- if the total number of facilities is less than $k - \theta_2$,
- if the facilities in the query do not include at least $k - \theta_2$ existing facilities of the client.

There is a tradeoff between utility and privacy in the selection of these threshold values. Selecting small θ_1 and θ_2 increases privacy, however, the utility of the

[1] θ_1 and θ_2 are design parameters of RNNQ, AVGQ, and MAXQ to be decided by the server.

protocols decreases due to rejection of more queries. Therefore, there cannot be an optimal threshold value for the protocols.

Moreover, when the query result includes a small number of users, the client can make an estimate about these users. For instance, there may be only one user whose nearest facility is a particular facility in RNNQ. Hence, if the RNN cardinality of a facility is one in RNNQ, the client can predict that user using its background knowledge. To prevent such privacy leaks in our protocols, we explain how to provide differential privacy in Section 3.4.

Finally, we also assume that during the protocol, communication is encrypted between the server and the client against an eavesdropper and that the server and the client(s) do not collude.

3.2 Server-based Query Processing Protocols

In this section, we propose server-based solutions that preserve the privacy while processing the queries in Section 3.1.2. Table 3.1 shows the symbols used in the protocols. The underlying protocols utilize the additive homomorphic property to hide sensitive data from other parties by calculating the sum of the encrypted values without decrypting them. We utilize Paillier cryptosystem as an additive homomorphic scheme satisfying $E(x) \cdot E(y) = E(x + y)$. In the server-based protocols, the server creates a public and private key pair (PK_s, SK_s), and shares the public key with the client. The client can encrypt any value or perform homomorphic operations on the ciphertexts, but only the server can decrypt encrypted messages. The server performs the majority of the encryptions in the protocols.

In the setup phase, the server generates (PK_s, SK_s) for Paillier cryptosystem. In addition, the server selects a superset $\mathcal{U} = \{U_1, ..., U_n\}$ of $\mathcal{U}_\mathcal{S}$ such that $\mathcal{U}_\mathcal{S} \subset \mathcal{U}$. The aim of selecting $\mathcal{U}$ is hiding $\mathcal{U}_\mathcal{S}$ (sensitive data 2(a)) from the client. For instance, let the identifier used in the protocols be mobile phone numbers.

Table 3.1: Notations used in Chapter 3.

m_s, m_c	modulus in Paillier generated by ($\mathcal{S}$, $\mathcal{C}$)
g_s, g_c	base in Paillier generated by ($\mathcal{S}$, $\mathcal{C}$)
PK_s, PK_c	public keys of $\mathcal{S}$ and $\mathcal{C}$
SK_s, SK_c	private keys of $\mathcal{S}$ and $\mathcal{C}$
$E_s(x)$, $E_c(x)$	Encryption of message x using (PK_s, PK_c)
$[x]_s$, $[x]_c$	denotes x is encrypted using (PK_s, PK_c)
$D_s([x]_s)$, $D_c([x]_c)$	Decryption of ciphertext x using (SK_s, SK_c)
$d(a, b)$	Distance between points a and b
$\mathcal{U}_\mathcal{S}, \mathcal{U}_\mathcal{C}$	user sets of $\mathcal{S}$ and $\mathcal{C}$
$\mathcal{U}$	superset of $\mathcal{U}_\mathcal{S}$ and $\mathcal{U}_\mathcal{C}$
$\mathcal{U}_\mathcal{I}$	$\mathcal{U}_\mathcal{S} \cap \mathcal{U}_\mathcal{C}$
n, n_s, n_c, n_I	total number of users in ($\mathcal{U}$, $\mathcal{U}_\mathcal{S}$, $\mathcal{U}_\mathcal{C}$, $\mathcal{U}_\mathcal{I}$)
$\mathcal{F}$	set of existing facilities of $\mathcal{C}$
k	total number of existing facilities
q, $\mathcal{Q}$	result (value, set) of the query
w	random number greater than q in MAXQ

Location-based service providers such as Foursquare and mobile telecommunication operators, and most businesses such as banks, hotels, and retailers typically know the mobile phone numbers of their customers. Hence, they can use mobile phone numbers as identifiers. Assume the phone numbers consist of 7 digits and there are 50 different mobile operator codes. When the superset $\mathcal{U}$ contains all possible mobile phone numbers, n becomes 500 million. Since $\mathcal{U}$ contains all possible numbers, it completely protects $\mathcal{U}_\mathcal{S}$ from the client. Another example is using national identification numbers as identifier. If national id numbers consist of 9 digits and the superset $\mathcal{U}$ contains all possible id numbers, n becomes one billion. The server shares $PK_s = (g_s, m_s)$ and $\mathcal{U}$ with the client. Note that all multiplications and exponentiations of ciphertexts in the server-based protocols are calculated in mod m_s^2.

Figure 3.2 shows the overview of the setup phase and the protocols. The server-based protocols consist of 10 steps. Steps 1, 4, 7, and 9 are the communication steps. In the first step, the client sends the query and the facility locations ($\mathcal{F}$) to the server. Step 2 is the calculation of distances between facilities and users. The server determines the nearest facility for each user. Since encrypted values

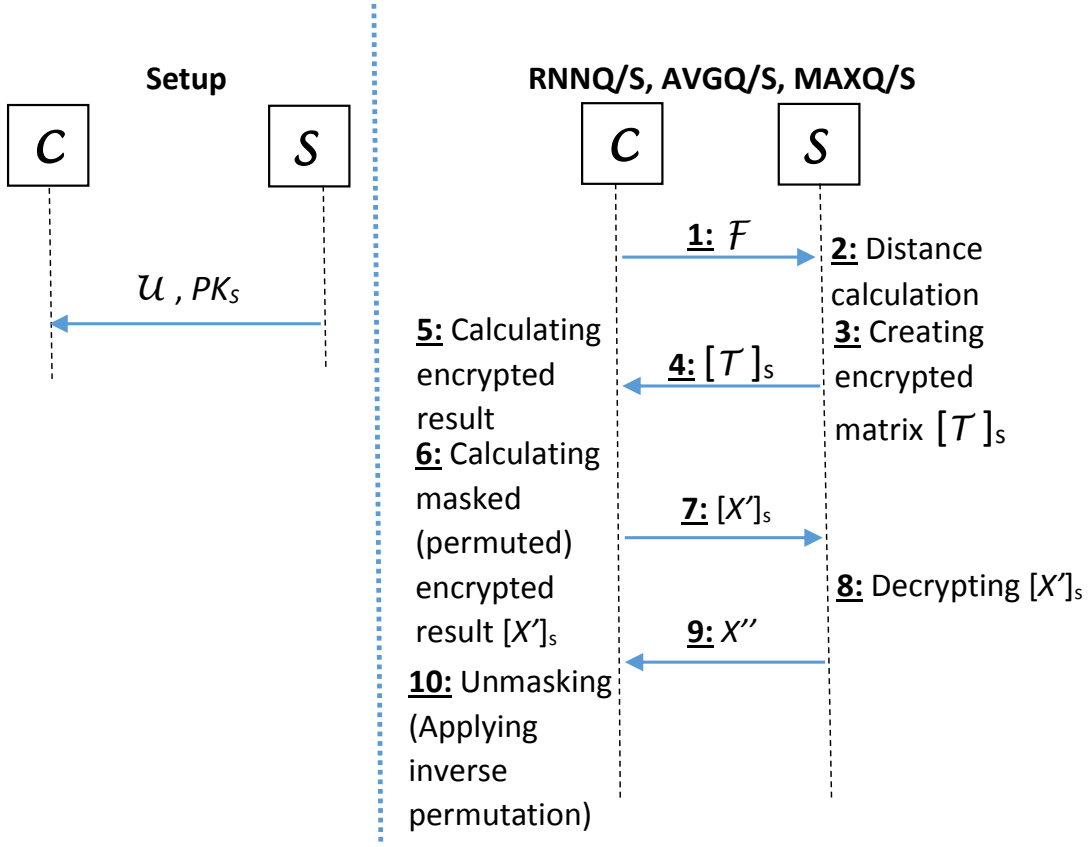


Figure 3.2: Overview of the server-based query processing protocols.

cannot be decrypted by the client, the server computes encrypted values based on nearest facility of each user in Step 3 to hide $\mathcal{U}_S$ and user locations (sensitive data 2(a) & 2(b)) from the client. Using the encrypted values, the client calculates the ciphertext of the query result by utilizing homomorphic properties of Paillier cryptosystem in Step 5. To hide $\mathcal{U}_C$ and the query result (sensitive data 1 & 3) from the server, the client masks the encrypted query result in Step 6 before sending to the server for decryption. The server decrypts the encrypted masked result in Step 8 and obtains the masked result. Due to masking in Step 6, the server cannot deduce the query result. In Step 10, the client applies unmasking and finds the query result.

3.2.1 RNN Cardinality Query (RNNQ/S)

Let q_i be the total number of users in $\mathcal{U}_\mathcal{I}$ whose nearest facility is F_i. This query returns the q_i values for each facility $F_i \in \mathcal{F}$. Hence, $\mathcal{Q} = \{q_1, ..., q_k\}$ is the query result.

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- **Step 2:** $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility. $\mathcal{S}$ aborts the protocol if it detects a threat as described in Section 3.1.3.
- **Step 3:** $\mathcal{S}$ sets $t_{i,j} = 1$ if $U_i \in \mathcal{U}_\mathcal{S}$ and F_j is the nearest facility of U_i; and $t_{i,j} = 0$ otherwise. $\mathcal{S}$ creates $n \times k$ matrix $[\mathcal{T}]_s$. $\mathcal{S}$ calculates $[T_{i,j}]_s = E_s(t_{i,j})$ for each element of matrix $[\mathcal{T}]_s$.
- **Step 4:** $\mathcal{S}$ sends encrypted matrix $[\mathcal{T}]_s$ to $\mathcal{C}$.
- **Step 5:** For each facility F_j, $\mathcal{C}$ calculates one ciphertext $[x_j]_s = \prod_{U_i \in \mathcal{U}_\mathcal{C}} [T_{i,j}]_s$. Hence, $[x_j]_s = E_s(q_j)$.
- **Step 6:** $\mathcal{C}$ selects k random values $\{v_1, ..., v_k\}$. $\mathcal{C}$ masks $[x_i]_s$ values with v_i values by calculating $[x'_i]_s = [x_i]_s \cdot E_s(v_i)$ for each $i \in \{1, 2, ..., k\}$. $[X']_s = \{[x'_1]_s, [x'_2]_s, ..., [x'_k]_s\}$.
- **Step 7:** $\mathcal{C}$ sends $[X']_s$ to $\mathcal{S}$.
- **Step 8:** $\mathcal{S}$ computes $x''_i = D_s([x'_i]_s)$ for each $i \in \{1, 2, ..., k\}$. $X'' = \{x''_1, x''_2, ..., x''_k\}$.
- **Step 9:** $\mathcal{S}$ sends X'' to $\mathcal{C}$.
- **Step 10:** Clearly, x''_i is equal to $q_i + v_i$ for each $i \in \{1, 2, ..., k\}$. $\mathcal{C}$ obtains $\mathcal{Q} = \{q_1, ..., q_k\}$ after unmasking.

We illustrate these steps with an example scenario. Let the identifier used by the server and the client consists of one digit, and id numbers of the users of the server be $1, 3, 5, 6, 7, 9$. The server can select the superset $\mathcal{U} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ such that $\mathcal{U}_\mathcal{S} \subset \mathcal{U}$. Assume that we have two facilities F_1 and F_2. When the client requests RNNQ/S, the server determines the nearest facility of its 6 users. Let F_1 be the nearest facility of the users 1, 6, and 9, F_2 be the nearest facility of the users 3, 5, and 7. In Step 3, the server computes $[T_1]_s = \{E_s(0), E_s(1), E_s(0), E_s(0), E_s(0), E_s(0), E_s(1), E_s(0), E_s(0), E_s(1)\}$ for F_1 and $[T_2]_s = \{E_s(0), E_s(0), E_s(0), E_s(1), E_s(0), E_s(1), E_s(0), E_s(1), E_s(0), E_s(0)\}$ for F_2. The server sends these encrypted values $[\mathcal{T}]_s$ to the client in Step 4. Let id numbers of the users of the client be $1, 2, 3, 5$. In Step 5, the client calculates two ciphertexts for two facilities by multiplying the ciphertexts of its users in $[\mathcal{T}]_s$. That is, $[x_1]_s = [T_{1,1}]_s \cdot [T_{1,2}]_s \cdot [T_{1,3}]_s \cdot [T_{1,5}]_s$ and $[x_2]_s = [T_{2,1}]_s \cdot [T_{2,2}]_s \cdot [T_{2,3}]_s \cdot [T_{2,5}]_s$. These values are the encryption of the query results such as $[x_1]_s = E_s(1)$ and $[x_2]_s = E_s(2)$. Let two random values selected by the client in Step 6 be 15 and 11. The client encrypts these random values and sends $[x'_1]_s = [x_1]_s \cdot E_s(15)$ and $[x'_2]_s = [x_2]_s \cdot E_s(11)$ to the server. The server decrypts these values in Step 8 and obtains $x''_1 = 16$ and $x''_2 = 13$. When the client receives these masked values, it subtracts the random values and obtains $q_1 = 1$ and $q_2 = 2$. Therefore, the client learns the RNN cardinality of F_1 and F_2.

3.2.2 Average Distance Query (AVGQ/S)

Let q be the average distance between users in $\mathcal{U}_\mathcal{I}$ and each one's nearest facility. The protocol is defined as follows:

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- **Step 2:** $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility. $\mathcal{S}$ aborts the protocol if it detects a threat as described in Section 3.1.3.

- **Step 3:** $\mathcal{S}$ creates $2 \times n$ matrix $[\mathcal{T}]_s$. For each user U_i in the superset $\mathcal{U}$, $\mathcal{S}$ calculates $[T_{1,i}]_s = E_s(d_i)$ and $[T_{2,i}]_s = E_s(1)$ if $U_i \in \mathcal{U}_\mathcal{S}$; and $[T_{1,i}]_s = E_s(0)$ and $[T_{2,i}]_s = E_s(0)$ otherwise (i.e. $U_i \notin \mathcal{U}_\mathcal{S}$).

- **Step 4:** $\mathcal{S}$ sends encrypted matrix $[\mathcal{T}]_s$ to $\mathcal{C}$.

- **Step 5:** $\mathcal{C}$ computes $[x_1]_s = \prod_{U_i \in \mathcal{U}_\mathcal{C}} [T_{1,i}]_s$ and $[x_2]_s = \prod_{U_i \in \mathcal{U}_\mathcal{C}} [T_{2,i}]_s$. $[x_1]_s$ is equal to $E_s(q \cdot n_I)$ and $[x_2]_s$ is equal to $E_s(n_I)$. Therefore, the query result (average distance) is equal to $\frac{[x_1]_s}{[x_2]_s}$.

- **Step 6:** $\mathcal{C}$ selects two random values v_1 and v_2. Then, $\mathcal{C}$ masks $[x_1]_s$ and $[x_2]_s$ by calculating $[x'_1]_s = [x_1]_s \cdot E_s(v_1)$ and $[x'_2]_s = [x_2]_s \cdot E_s(v_2)$. $[X']_s = \{[x'_1]_s, [x'_2]_s\}$.

- **Step 7:** $\mathcal{C}$ sends $[X']_s$ to $\mathcal{S}$.

- **Step 8:** $\mathcal{S}$ computes $x''_1 = D_s([x'_1]_s)$ and $x''_2 = D_s([x'_2]_s)$. $X'' = \{x''_1, x''_2\}$.

- **Step 9:** $\mathcal{S}$ sends X'' to $\mathcal{C}$.

- **Step 10:** Clearly, x''_1 and x''_2 are equal to $q \cdot n_I + v_1$ and $n_I + v_2$, respectively. $\mathcal{C}$ obtains q after the unmasking and division operations.

3.2.3 Maximum Distance Query (MAXQ/S)

Let q be the maximum distance between a user in $\mathcal{U}_\mathcal{I}$ and her nearest facility. The protocol is defined as follows:

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.

- **Step 2:** $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility. $\mathcal{S}$ aborts the protocol if it detects a threat as described in Section 3.1.3.

- **Step 3:** Let max be the maximum distance between a user in $\mathcal{U}_{\mathcal{S}}$ and her nearest facility. $\mathcal{S}$ selects a value w, which is greater than max. $\mathcal{S}$ creates $n \times w$ matrix $[\mathcal{T}]_s$. For each $j \in \{1, .., w\}$, $\mathcal{S}$ sets $[T_{i,j}]_s = E_s(1)$ if $U_i \in \mathcal{U}_{\mathcal{S}}$ and $d_i = j$; and $[T_{i,j}]_s = E_s(0)$ otherwise.

- **Step 4:** $\mathcal{S}$ sends encrypted matrix $[\mathcal{T}]_s$ to $\mathcal{C}$.

- **Step 5:** For each $j \in \{1, .., w\}$, $\mathcal{C}$ calculates one ciphertext $[x_j]_s = \prod_{U_i \in \mathcal{U}_{\mathcal{C}}} [T_{i,j}]_s$. Therefore, $[x_j]_s$ is equal to the encryption of the total number of users in $\mathcal{U}_{\mathcal{I}}$ whose distance to the nearest facility is equal to j. The query result q is equal to the maximum j value such that $D_s([x_j]_s) \neq 0$.

- **Step 6:** $\mathcal{C}$ selects w random values $\{v_1, ..., v_w\}$ for masking $[x_i]_s$ values. Then, $\mathcal{C}$ calculates $[x_i]_s^{v_i}$ for each $i \in \{1, 2, ..., w\}$. If $[x_i]_s$ is the encryption of 0, $[x_i]_s^{v_i}$ is the encryption of 0. Therefore, q is still equal to the maximum j value such that $D_s([x_j]_s^{v_j}) \neq 0$. To protect query result from $\mathcal{S}$, $\mathcal{C}$ selects a random permutation π and calculates $[X']_s = \{[x'_1]_s, ..., [x'_w]_s\} = \pi(\{[x_1]_s^{v_1}, ..., [x_w]_s^{v_w}\})$.

- **Step 7:** $\mathcal{C}$ sends $[X']_s$ to $\mathcal{S}$.

- **Step 8:** For each $i \in \{1, 2, ..., w\}$, $\mathcal{S}$ sets $x''_i = 1$ if $D_s([x'_i]_s) \neq 0$, and $x''_i = 0$ if $D_s([x'_i]_s) = 0$. $X'' = \{x''_1, ..., x''_w\}$.

- **Step 9:** $\mathcal{S}$ sends X'' to $\mathcal{C}$.

- **Step 10:** $\mathcal{C}$ applies the inverse permutation π^{-1} and obtains $\{x'''_1, ..., x'''_w\} = \pi^{-1}(\{x''_1, ..., x''_w\})$. The maximum j value such that $x'''_j = 1$ is the result of the query.

3.2.4 Security Analysis of Server-Based Protocols

In this section, we prove the security of the server-based protocols in the semi-honest model. Semi-honest parties follow the protocol correctly; however, they may try to learn additional information by analyzing the messages they receive throughout the protocol. In general, in secure two-party protocol, the goal of

the parties is to compute a desired output pair $f(x, y) = (f_1(x, y), f_2(x, y))$ from their inputs x and y without revealing them to each other. The first party wants to obtain $f_1(x, y)$ and the second party wants to obtain $f_2(x, y)$ at the end of the protocol. During the protocol, the view of a party consists of its input, its random-tape, and sequence of incoming messages throughout the protocol. A protocol privately computes $f(x, y)$ if a party's view can be simulated from its input and output [48].

More formally, let Π be a secure two-party protocol for computing $f(x, y)$. The views of the parties are denoted as $\text{VIEW}_1^{\Pi}(x, y)$ and $\text{VIEW}_2^{\Pi}(x, y)$. Then, the security of a deterministic protocol in semi-honest model is defined as follows [48]:

Definition 3 *The protocol* Π *privately computes* $f(x, y)$ *if there exist probabilistic polynomial-time simulators* Sim_1 *and* Sim_2 *such that*

$$\{Sim_1(x, f_1(x, y))\} \stackrel{c}{\equiv} \left\{\text{VIEW}_1^{\Pi}(x, y)\right\}$$
$$\{Sim_2(x, f_2(x, y))\} \stackrel{c}{\equiv} \left\{\text{VIEW}_2^{\Pi}(x, y)\right\}$$

where $\stackrel{c}{\equiv}$ implies computational indistinguishability. Therefore, a party's privacy is guaranteed if there exists a simulator that can generate a view indistinguishable from the view of the opposing party. In the following, we prove the security of the server-based protocols using this simulation paradigm.

Let the client be the first party and the server be the second party in our protocols. The private input x of the client is $\mathcal{U}_{\mathcal{C}}$ and the private input y of the server is $\mathcal{U}_{\mathcal{S}}$ and the user locations. $\mathcal{F}$ is also the input of the protocol, which is commonly known by the server and the client. As discussed in Section 3.1.3, it should not be hidden from the server to prevent attacks via exhaustive client queries. In addition, PK_s, PK_c, and $\mathcal{U}$ are also known by the server and the client as background information. As discussed before, $\mathcal{U}$ is the superset of the users for keeping the user list of parties from each other. The client should get **query result** as $f_1(x, y)$ at the end of the protocol while the server receives no output (i.e. $f_2(x, y) = \perp$).

Since the steps of the server-based protocols are similar as shown in Figure 3.2, we consider RNNQ/S in the proof. The security of the other protocols can be proved similarly. In RNNQ/S, $\mathcal{Q} = \{q_1, ..., q_k\}$ is the query result where q_i is the total number of users in $\mathcal{U}_\mathcal{I}$ whose nearest facility is F_i. Therefore, the view of the client (VIEW_1) consists of $\mathcal{U}_\mathcal{C}$, $\mathcal{F}$, $[\mathcal{T}]_s$, and $\mathcal{Q}$. To prove that the server's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_1 such that $Sim_1(\mathcal{U}_\mathcal{C}, \mathcal{F}, \mathcal{Q})$ is computationally indistinguishable from VIEW_1. Since $[\mathcal{T}]_s$ contains $n \cdot k$ Paillier ciphertexts, Sim_1 can generate $n \cdot k$ random numbers between 0 and m_s^2 and these numbers are computationally indistinguishable from the ciphertexts in $[\mathcal{T}]_s$ due to the semantic security of Paillier cryptosystem.

On the other hand, the view of the server (VIEW_2) consists of $\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F}$, and X''. To prove that the client's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_2 such that $Sim_2(\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F})$ is computationally indistinguishable from VIEW_2. This is satisfied by letting Sim_2 generate k random numbers between 0 and m_s to simulate X'' because X'' contains k values $\{q_1 + v_1, ..., q_k + v_k\}$ where each v_i is a randomly selected number by the client. Thus, we conclude that RNNQ/S protocol securely processes RNN Cardinality queries in semi-honest model.

Although the server-based protocols preserve privacy in semi-honest model, they can be vulnerable to the attack of a malicious client. A malicious client can calculate the encrypted result in Step 5 for a specific customer U_i. Therefore, the client can obtain information about the location of U_i such as the nearest facility of U_i and its distance to the nearest facility. However, in any case, it is not possible to find the exact location of U_i. To prevent the defined attack by malicious clients while providing the exact query result, we propose client-based protocols in Section 3.3. Moreover, Section 3.4 explains satisfying differential privacy in server-based protocols. To protect the privacy of individuals from these kinds of attacks, differential privacy gives a guarantee that presence or absence of an individual will not affect the final output of the algorithm significantly. When the queries return noisy results instead of exact results, a malicious client cannot

obtain the nearest facility of a specific user U_i and its distance to the nearest facility. For instance, let F_1 be the nearest facility of U_i. Then, the exact query result is $(1, 0, 0, ..., 0)$ for the defined attack. However, adding a noise to each of these values will prevent the information leak about U_i. Therefore, differential privacy provides privacy guarantees against such attacks from the malicious client.

3.3 Client-based Query Processing Protocols

In the protocols defined in Section 3.2, the data is encrypted with the public key of the server. The server computes most of the encryptions, which dominates the computation cost. In this section, we propose protocols using the public and private keys (PK_c, SK_c) of the client, where the client computes the majority of the encryptions, however, instead of performing encryptions during each query, the client performs encryptions in the setup. This makes the setup phase of these protocols more costly than the protocols in Section 3.2, however, query processing in these protocols is more efficient in terms of computation and communication costs. The protocols defined in this section also return exact query results as in Section 3.2. We describe achieving differential privacy during the client-based protocols in Section 3.4.

In the setup phase, the client generates a public and private key pair (PK_c, SK_c) for Paillier cryptosystem. The client shares $PK_c = (g_c, m_c)$ with the server. All multiplications and exponentiations of ciphertexts in the client-based protocols are calculated in mod m_c^2. In addition, the server selects a superset $\mathcal{U} = \{U_1, ..., U_n\}$ and shares with the client, as described in Section 3.2. Then, for each $U_i \in \mathcal{U}$, the client calculates $[T_i]_c = E_c(0)$ if $U_i \notin \mathcal{U}_\mathcal{C}$ and $[T_i]_c = E_c(1)$ if $U_i \in \mathcal{U}_\mathcal{C}$. The client sends $[\mathcal{T}]_c = \{[T_1]_c, ..., [T_n]_c\}$ to the server. Let r_i be the random number used in the calculation of $[T_i]_c$. To prevent malicious client attack described in Section 3.2.4, the client sends the total number of its users (n_c) and $r = \prod_{i=1}^{n} r_i$ to the server. The server multiplies all $[T_i]_c$ values and obtains a ciphertext which should be equal to encryption of n_c. That is,

$E_c(n_c) = g_c^{n_c} \cdot r^m = \prod_{i=1}^{n} [T_i]_c \pmod{m_c^2}$. The server encrypts n_c with the random value r and verifies the total number of the client's users. If $\prod_{i=1}^{n} [T_i]_c$ is not equal to $E_c(n_c)$ or n_c is less than a threshold value, the server aborts the protocol. Therefore, a malicious client cannot get the query result for a specific user.

Once the client sends n ciphertexts to the server, any of the aforementioned queries can be performed with small computation and communication overheads. We can assume that the users of the client do not change frequently. Small number of changes on the user list do not have a notable effect on query results as well. Hence, the client can update the encrypted list $[\mathcal{T}]_c$, when there is a significant change on its user list. In addition, when the client decides an update in $[\mathcal{T}]_c$, it is not necessary to update all values in $[\mathcal{T}]_c$. The client can only update a subset of users that contains the users to be changed. For instance, if the superset $\mathcal{U}$ includes 100 million users, to change 100 users in $[\mathcal{T}]_c$, the client can update a subset of $[\mathcal{T}]_c$ containing one million users instead of all users in $[\mathcal{T}]_c$.

Figure 3.3 shows the overview of the setup phase and the protocols. The protocols in this section consist of 6 steps. The server and the client communicate in Steps 1 and 5. Step 2 is the calculation of distances as in server-based protocols. In Step 3, the server utilizes homomorphic properties of Paillier cryptosystem to calculate the encryption of the query result by using encrypted values in $[\mathcal{T}]_c$. Before sending the encrypted result to the client, the server anonymizes the result by multiplying it with the encryption of zero in Step 4. This multiplication does not alter the result; it only prevents the server from tracking users by the client. Therefore, the server hides user locations from the client. In Step 6, the client obtains the query result after decryption. Since the server only receives the locations of the facilities during query processing, it is not possible for the server to determine query result.

3.3.1 RNN Cardinality Query (RNNQ/C)

$\mathcal{Q} = \{q_1, ..., q_k\}$ is the query result. Figure 3.4 illustrates the steps of the protocol for the same example scenario explained in Section 3.2.1. The protocol is defined

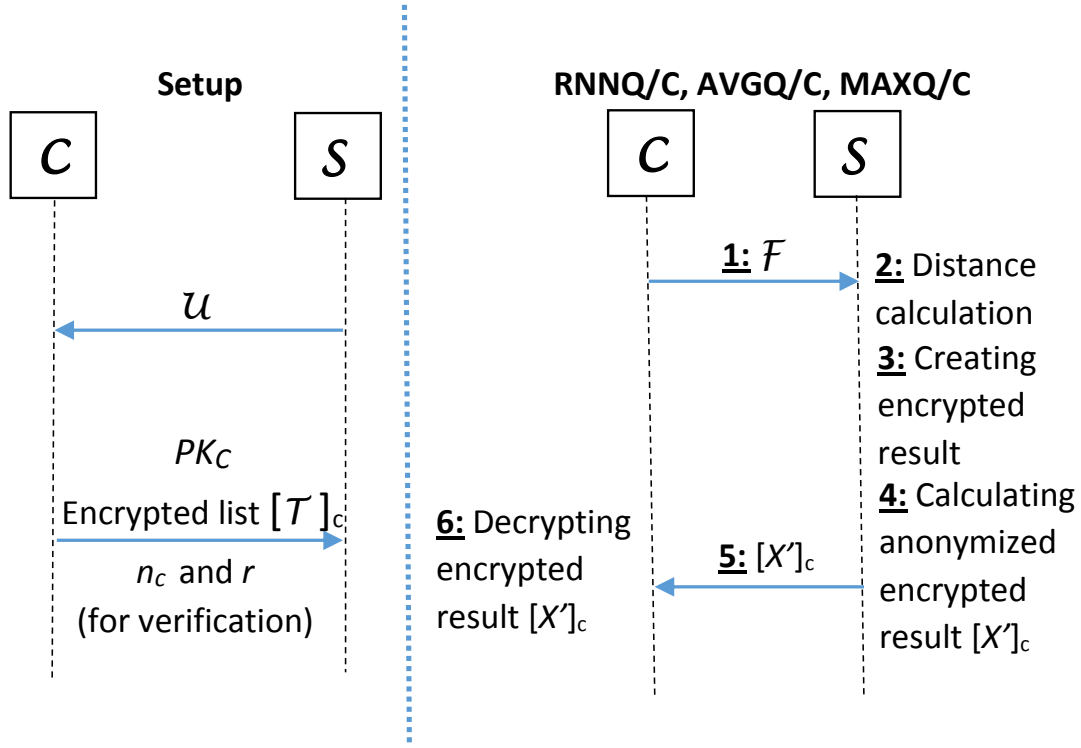


Figure 3.3: Overview of the client-based query processing protocols.

as follows:

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- **Step 2:** $\mathcal{S}$ checks the facility locations and aborts the protocol if $\mathcal{C}$ adds more than θ_1 new facilities or removes more than θ_2 existing facilities as described in Section 3.1.3. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$.
- **Step 3:** For each facility F_j, $\mathcal{S}$ calculates the $[x_j]_c$ value by multiplying $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$ and the nearest facility of U_i is F_j. At the end of this step, $\mathcal{S}$ forms $[X]_c = \{[x_1]_c, ..., [x_k]_c\}$ where $[x_i]_c$ is the encryption of q_i. In this step, $\mathcal{S}$ computes the encrypted result.
- **Step 4:** $\mathcal{S}$ encrypts 0 using k different random values and calculates $[x'_i]_c = [x_i]_c \cdot E_c(0)$ for each $i \in \{1, ..., k\}$.

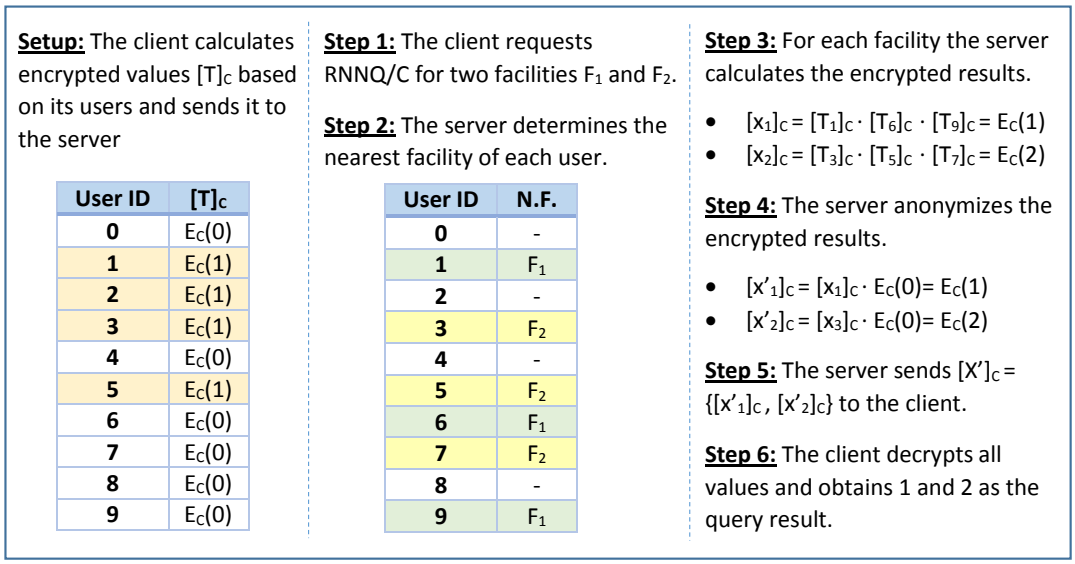


Figure 3.4: An example scenario for RNNQ/C protocol.

- **Step 5:** $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, ..., [x'_k]_c\}$ to $\mathcal{C}$.
- **Step 6:** $\mathcal{C}$ decrypts all $[x'_i]_c$ values in $[X']_c$, and clearly, $D_c([x'_i]_c)$ is equal to q_i. $\mathcal{C}$ obtains $\mathcal{Q} = \{q_1, ..., q_k\}$.

3.3.2 Average Distance Query (AVGQ/C)

The average distance between users in $\mathcal{U}_\mathcal{I}$ and each one's nearest facility is q. The protocol is defined as follows:

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- **Step 2:** As described in RNNQ/C protocol, the server aborts the protocol if it detects a threat. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility.
- **Step 3:** $\mathcal{S}$ calculates the multiplication of $[T_i]_c^{d_i}$ values and the multiplication of $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$. That is, $[x_1]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S}} [T_i]_c^{d_i}$ and

$[x_2]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S}} [T_i]_c$. Clearly, $[x_1]_c$ is equal to $E_c(q \cdot n_I)$ and $[x_2]_c$ is equal to $E_c(n_I)$.

- **Step 4:** $\mathcal{S}$ calculates $[x'_1]_c = [x_1]_c \cdot E_c(0)$ and $[x'_2]_c = [x_2]_c \cdot E_c(0)$.
- **Step 5:** $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, [x'_2]_c\}$ to $\mathcal{C}$.
- **Step 6:** $\mathcal{C}$ decrypts $[x'_1]_c$ and $[x'_2]_c$. Clearly, $D_c([x'_1]_c)$ is equal to $q \cdot n_I$ and $D_c([x'_2]_c)$ is equal to n_I. $\mathcal{C}$ obtains q after division.

3.3.3 Maximum Distance Query (MAXQ/C)

The maximum distance between a user in $\mathcal{U}_\mathcal{I}$ and her nearest facility is q. The protocol is defined as follows:

- **Step 1:** $\mathcal{C}$ sends the location of each facility to $\mathcal{S}$.
- **Step 2:** As described in RNNQ/C protocol, the server aborts the protocol if it detects a threat. $\mathcal{S}$ calculates the distance between each facility and each user in $\mathcal{U}_\mathcal{S}$. $\mathcal{S}$ determines the nearest facility of each user U_i in $\mathcal{U}_\mathcal{S}$ and the distance d_i to the nearest facility. Let max be the maximum distance between a user in $\mathcal{U}_\mathcal{S}$ and her nearest facility. $\mathcal{S}$ selects a value w, which is greater than max.
- **Step 3:** For each $j \in \{1, .., w\}$, $\mathcal{S}$ calculates the multiplication of $[T_i]_c$ values such that $U_i \in \mathcal{U}_\mathcal{S}$ and $d_i = j$. That is, $\mathcal{S}$ computes $[x_j]_c = \prod_{U_i \in \mathcal{U}_\mathcal{S} \& d_i = j} [T_i]_c$. If there is no such U_i, $\mathcal{S}$ sets $[x_j]_c = E_c(0)$. Therefore, $[x_j]_c$ is equal to the encryption of the total number of users in $\mathcal{U}_\mathcal{I}$ whose distance to the nearest facility is equal to j. The query result q is equal to the maximum j value such that $D_c([x_j]_c) \neq 0$.
- **Step 4:** At the end of the protocol, $\mathcal{C}$ should not learn anything more than the query result. To hide the $[x_j]_c$ values from $\mathcal{C}$, $\mathcal{S}$ randomizes the $[x_j]_c$ values by exponentiation. $\mathcal{S}$ selects w random values $\{v_1, ..., v_w\}$. Then, $\mathcal{S}$ calculates $[x'_i]_c = [x_i]_c^{v_i}$ for each $i \in \{1, 2, ..., w\}$. If $[x_i]_c$ is the encryption of

0, $[x'_i]_c$ is the encryption of 0. Therefore, q is still equal to the maximum j value such that $D_c(\left[x'_j\right]_c) \neq 0$.

- **Step 5:** $\mathcal{S}$ sends $[X']_c = \{[x'_1]_c, ..., [x'_w]_c\}$ to $\mathcal{C}$.
- **Step 6:** $\mathcal{C}$ decrypts all $[x'_i]_c$ values. $\mathcal{C}$ obtains q, since it is equal to the maximum j value such that $D_c(\left[x'_j\right]_c) \neq 0$.

3.3.4 Security Analysis of Client-Based Protocols

In this section, we prove the security of the client-based protocols using the simulation paradigm described in Section 3.2.4. To prove the security of the protocols we need to show that there exists two probabilistic polynomial-time simulators Sim_1 and Sim_2 for simulating the views of the client and the server, respectively. In client-based protocols, the view of the client only consists of its input and output. The server only sends the encrypted query result to the client in Step 5. Since, the encrypted result is anonymized in Step 4, $[X']_c$ does not contain any information about the users. Therefore, the view of the client can obviously be simulated by Sim_1 and the privacy of the server is assured.

The view of the server (VIEW_2) consists of $\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F}$, and $[\mathcal{T}]_c$. To prove that the client's privacy is assured in the protocol, we need to show that there exists a probabilistic polynomial-time simulator Sim_2 such that $Sim_2(\mathcal{U}_\mathcal{S}$, user locations, $\mathcal{F}) \stackrel{c}{\equiv} \text{VIEW}_2$. This is satisfied by letting Sim_2 generate n random numbers between 0 and m_c^2. These numbers are computationally indistinguishable from the ciphertexts in $[\mathcal{T}]_c$ due to the semantic security of Paillier cryptosystem. Hence, we conclude that the client-based protocols privately process the queries in semi-honest model.

3.4 Protocols with Differential Privacy

Differential privacy is a framework to formalize privacy in statistical databases. The security proofs indicate that the proposed protocols reveal no more information than the output of the queries. However, providing aggregate statistical information about a database may reveal information about the individuals in the dataset. All of the queries (RNNQ, AVGQ, and MAXQ) that are studied in this chapter return aggregate results and these query results may cause information leaks in some cases. For example, RNNQ returns the cardinality of $RNN(F_i)$ for each facility F_i in $\mathcal{F}$. If the RNN cardinality of a facility is 1, this user can be predicted with background knowledge. However, only a region containing the user's location can be inferred. In any case, it is not possible to find the exact location of a user.

The protocols defined in Section 3.2 and Section 3.3 return exact query results. To achieve differential privacy in these protocols, we need to add controlled random noise to the query result. As discussed in Section 2.3, one needs to define the sensitivity of a query to determine the amount of noise to be added to the result of a query. Now, we show the sensitivity of each considered query and how to add the noise during the protocols.

RNNQ returns the total number of users attracted by each facility. It can be thought as a histogram query [26] and its sensitivity is 2. When there is a single change in the database, RNN of at most two facilities may change. Therefore, we add a noise $Laplace(\frac{2}{\epsilon})$ to the RNN cardinality of each facility.

AVGQ returns two values: (i) the total number of users in $\mathcal{U}_\mathcal{I}$ (n_I) and (ii) the total distance between each user and her nearest facility ($q \cdot n_I$). Thus, we need to calculate the sensitivity for both subqueries. Since the total number of users is a counting query, the sensitivity for n_I is 1. For the total distance, the sensitivity is the maximum distance (max) between a user in $\mathcal{U}_\mathcal{S}$ and her nearest facility. Therefore, we add the noise from $Laplace(\frac{1}{\epsilon})$ to n_I and $Laplace(\frac{max}{\epsilon})$ to $q \cdot n_I$.

MAXQ returns w^2 values containing zero and non-zero elements. The largest index of a non-zero element is the result of the query. In MAXQ each of w values can be considered as a counting query, and hence the sensitivity of each one is 1. Therefore, we add $Laplace(\frac{1}{\epsilon})$ to each w values.

In the server-based protocols, the server adds noise to the query result in Step 8. Before sending the masked result X'' to the client, the server adds noise to the masked result. When the client applies unmasking in Step 10, it obtains the noisy result instead of the exact result.

In the client-based protocols, the server adds noise to the query result in Step 4. Before sending the encrypted result to the client, the server anonymizes the result by multiplying it with the encryption of zero in the client-based protocols. Instead of encrypting zero values, the server encrypts the values drawn from the Laplace distribution and multiplies the encryption of the noise with the encrypted query result. Due to homomorphic properties of Paillier cryptosystem, the noise will be added to the query result in plaintext. When the client decrypts query result in Step 6, it obtains the noisy result instead of the exact result.

3.5 Evaluation

In this section, we analyze the complexity, performance, and the utility of the proposed protocols. As there is no existing work that solves the stated problems, we only show the feasibility of our solutions. Firstly, we analyze the computation complexity and the communication costs theoretically in Section 3.5.1. In Section 3.5.2, we present the experimental efficiency evaluation of each protocol with respect to different parameters. In Section 3.5.3, we show the utility of the protocols when differential privacy is achieved.

[2] w is a random number that is selected by the server in the MAXQ/S and MAXQ/C protocols

Table 3.2: Computation performed in the proposed query processing protocols.

	$\mathcal{S}$	$\mathcal{C}$
RNNQ/S	$n_s \cdot k$ dist. $n \cdot k$ enc. k dec.	$n_c \cdot k$ mult. k enc.
AVGQ/S	$n_s \cdot k$ dist. $2 \cdot n$ enc. 2 dec.	$2 \cdot n_c$ mult. 2 enc. 1 div.
MAXQ/S	$n_s \cdot k$ dist. $n \cdot w$ enc. w dec.	$w \cdot (n_c - 1)$ mult. w exp. 2 per.
RNNQ/C	$n_s \cdot k$ dist. k enc. $n_s + k$ mult.	k dec.
AVGQ/C	$n_s \cdot k$ dist. 2 enc. $2 \cdot n_s$ mult. n_s exp.	2 dec. 1 div.
MAXQ/C	$n_s \cdot k$ dist. n_s mult. w exp. $\leq w$ enc.	$w - q + 1$ dec.

3.5.1 Complexity Analysis

In this section, we analyze the computation and communication costs of the proposed protocols in Section 3.2 and Section 3.3. Achieving differential privacy as described in Section 3.4 does not change the communication costs of the protocols. Moreover, its effect on computation time is negligible because only overhead to achieve differential privacy is producing the noise drawn from the Laplace distribution. Therefore, we give the computation costs of the protocols as described in Section 3.2 and Section 3.3.

Server-based protocols. Table 3.2 shows the total number of operations performed during server-based protocols in terms of total number of encryptions, decryptions, multiplications, exponentiations, distance calculations, and permutations. In all protocols, encryptions dominate the computation times. The

number of encryptions is proportional to n and the server performs at least n encryptions in each query. However, the server encrypts 0 or 1 in each encryption and it can encrypt these values offline before the protocol. When the server uses precomputed $E_s(0)$ and $E_s(1)$ values in these protocols, computation cost reduces significantly. In addition, all of these ciphertexts must be transferred to the client in each query. Hence, the computation costs of RNNQ/S, AVGQ/S, and MAXQ/S are $n \cdot k$, $2 \cdot n$, and $n \cdot w$ ciphertexts, respectively.

Client-based protocols. In the setup phase of the client-based protocols, n encryptions are computed by the client. The client sends these n ciphertexts to the server in the setup. Therefore, the communication overhead of the setup is n ciphertexts. After completion of the setup phase, all queries can be processed with small computation and communication overheads. Table 3.2 shows computation costs of client-based protocols in each query. Total number of encryptions in each query is very small with respect to the server-based protocols. The computation costs of RNNQ/C, AVGQ/C, and MAXQ/C are k, 2, and w ciphertexts, respectively.

3.5.2 Efficiency

We have implemented the protocols in Java and we used the implementation in [50] for Paillier cryposystem. All experiments were performed on a 64-bit Windows 7 machine with 2.6 GHz Intel Core i5 processor and 4 GB of RAM. We used 1024-bit modulus m_s and m_c in our tests and each ciphertext consists of 2048 bits. All distances were calculated by the server in the Euclidean metric.

In our experiments, we used real datasets [51] containing 227,428 check-ins in New York City and 573,703 check-ins in Tokyo. The x and y coordinates were scaled to integer values from 1 to 10,000. Since the total number of users in the datasets is less than 5,000, we considered each check-in location as the location of a separate user. Therefore, $n_s = 227,428$ in NYC dataset and $n_s = 573,703$ in Tokyo dataset. We randomly chose 20% of them as the users of the client. For existing facilities, we used the locations of 20 restaurants of a fast food chain in

New York and 10 restaurants of a fast food chain in Tokyo.

For synthetic datasets, the x and y coordinates of the user locations and facility locations were selected randomly as integer values from 1 to $maxCoordinate$. The user id values in the superset $\mathcal{U}$ were selected as the numbers from 1 to n. We randomly chose n_s of them as the users of the server and n_c of them as the users of the client. The key parameters in the implementation were n, n_s, n_c, n_I, k, $maxCoordinate$, m_s, and m_c (introduced in Table 3.1). w is another parameter in the Maximum Distance Query, which depends on the value of $maxCoordinate$. We present the experimental evaluation of the server-based protocols in Section 3.5.2.1 and the client-based protocols in Section 3.5.2.2.

3.5.2.1 Server-based Protocols

When we use 1024-bit m_s in Paillier encryption, one million encryption nearly takes 2 hours and 45 minutes and the size of one million ciphertexts is 250 MB. For the protocols RNNQ/S, AVGQ/S, and MAXQ/S, the computation times and communication costs are directly proportional to $n \cdot k$, $2 \cdot n$, and $n \cdot w$, respectively. Therefore, when n is one million, the computation time of each protocol is more than 2 hours and 45 minutes. Moreover, when n is one million, the amount of data exchanged during each protocol is more than 250 MB.

In our experiments, we set $n = 1,000,000$, $n_s = 100,000$, $n_c = 20,000$, $k = 25$, and $maxCoordinate = 10,000$ in the synthetic dataset. Running time of RNNQ/S with these parameters is high because it requires $n \cdot k$ encryptions for the encrypted matrix $[\mathcal{T}]_s$. However, all of these ciphertexts in $[\mathcal{T}]_s$ are either the encryption of 0 or the encryption of 1. Therefore, the encrypted values in $[\mathcal{T}]_s$ can be computed offline by the server. When the server precomputes $E_s(0)$ and $E_s(1)$ values before the protocol, the remaining computation takes 10 seconds for these parameters. For the NYC and Tokyo datasets, the query takes 20 seconds and 25 seconds, respectively. Similarly, for the synthetic dateset with given parameters, AVGQ/S takes 13.5 seconds, when the server computes $E_s(0)$ and $E_s(1)$ values before the protocol. Since the computation time of AVGQ/S is

directly proportional to n_s, the query takes nearly 35 and 70 seconds for the NYC and Tokyo datasets, respectively. The computation time of MAXQ/S mostly depends on the value of w, which is a randomly selected number by the server. The server performs w decryptions and the client performs w exponentiations and $w \cdot (n_c - 1)$ multiplications. For instance, when w is selected as 500, the computation time of MAXQ/S is nearly 5 minutes and it increases linearly when w increases. When w remains same, we observed the similar computation times for the real datasets.

Our experimental results show that the computation at the client's side is low in server-based query processing protocols. Step 3 of these protocols necessitates calculating an encrypted matrix $[\mathcal{T}]_s$. This step dominates the computation time of server-based protocols. However, the encrypted values can be computed offline by the server. To do offline computation, the server does not need to know the facility locations. The server can compute $E_s(0)$ and $E_s(1)$ values before the protocol. When the client sends the facility locations in a protocol, the server uses previously computed ciphertexts in the encrypted matrix $[\mathcal{T}]_s$. Hence, if the server computes these encryptions offline before the protocol, the remaining computation takes a few minutes on a single computer for millions of users. In addition, the computation time of calculating $[\mathcal{T}]_s$ can be reduced via parallel computations because all encryptions are independent. Server-based protocols can be preferable when the client cannot afford to perform the computations or when the client wants to outsource all the computations to the server. However, as we have shown, the encrypted values in $[\mathcal{T}]_s$ must be transferred to the client in each query.

3.5.2.2 Client-based Protocols

In Section 3.5.1, the computation complexity of each client-based protocol is given. When the client performs several queries, some of these computations are common. For instance, the server calculates the distance between each facility and each user in each query. For the same facilities in separate queries, the server does not need to calculate the same distance values. In our system model, the

locations of the existing facilities are considered as public and known by the server. The client can share these locations in the setup phase. Since the server knows the locations of the existing facilities, we assume that all the distances between users and existing facilities were calculated and the nearest facility of each user was determined by the server before the execution of protocols. In each query, the client sends a possible location for adding a new facility and the server only calculates the distance between the new location and each user. The server only updates the nearest facilities of the users who are attracted by the new facility. Therefore, we evaluate the following for each protocol under different parameter settings:

- **Precomputation time.** Most of the computation given in Table 3.2 can be precomputed by the server because the locations of the existing facilities are known by the server. Hence, we evaluate the precomputation time of each protocol separately.
- **Query processing time.** Once the server completes the precomputation, processing of each query requires low computation overhead. We evaluate the query processing time when the client sends a possible location for adding a new facility.
- **Amortized computation time.** When the client requests n_q queries, amortized computation time of a query is equal to ((precomputation time) + n_q· (query processing time)) / n_q.

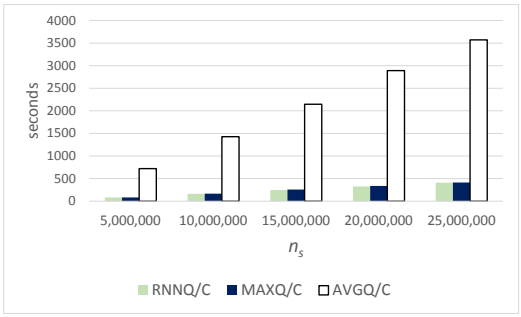
In the setup phase of the client-based protocols, the client computes n ciphertexts and shares them with the server. Therefore, the computation cost and the communication cost of the setup phase of the client-based protocols are directly proportional to n. One ciphertext consists of 2048 bits and one encryption takes 10 ms on the machine mentioned above. Therefore, if n is one million, the execution time of the setup phase is nearly 2 hours and 45 minutes[3] and the amount of data sent by the client to the server is 250 MB.

[3]Computation time can be further reduced via parallel computations.

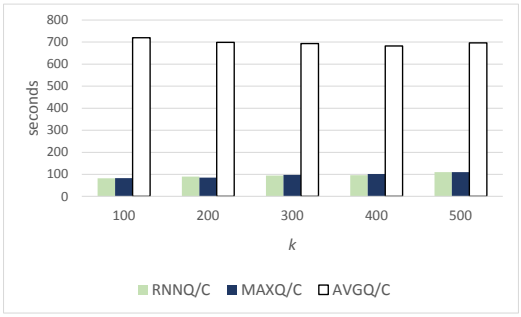
Table 3.2 shows computation costs of client-based protocols including precomputation and query processing. We evaluate the performance of the protocols with respect to n, n_s, n_c, n_I, k, and $maxCoordinate$. As evident in Table 3.2, the parameters n, n_c, and n_I have no effect on query processing times of the protocols. In our experiments, we observed the similar computation times for different values of these parameters. Therefore, increasing one of these parameters does not change the precomputation time and the query processing time of client-based protocols. For the other parameters n_s, k, and $maxCoordinate$, we analyze their effects on the precomputation time, the query processing time and the communication cost of each client-based protocol. In our experiments, we set $n = 200,000,000$, $n_s = 5,000,000$, $n_c = 1,000,000$, $n_I = 500,000$, $k = 100$, and $maxCoordinate = 10,000$, unless stated otherwise.

RNNQ/C. Table 3.2 shows the computation cost of the protocol, where n_s and k are the determining parameters. In this protocol, $n_s \cdot k$ distance calculations, n_s multiplications and k encryptions can be precomputed by the server. The client computes $[X]_c = \{[x_1]_c, ..., [x_k]_c\}$ before the protocol. When the client requests a query for a new facility location (F_{k+1}), the server only calculates the distance between F_{k+1} and each user. Then, the client multiplies the $[T_i]_c$ values of the users whose nearest facility is F_{k+1} and calculates $[x_{k+1}]_c$. The client also multiplies the inverse of $[T_i]_c$ values of the same users with the x_j values of their previous nearest neighbors. Therefore, during query processing the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, and nearly $\frac{n_s}{k}$ modular inverse calculations. The client also performs k decryptions during query processing.

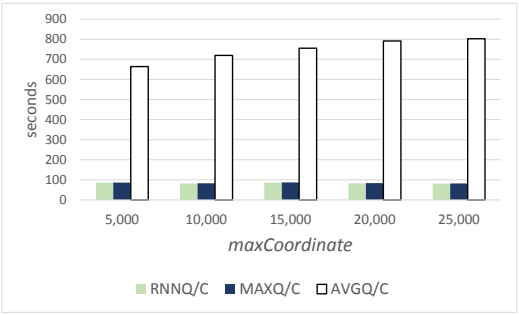
Although encryption and decryption are more expensive operations than multiplication, the most time consuming part in the precomputation time is n_s multiplications because n_s is much higher than k in our experiments. Therefore, the precomputation time mostly depends on n_s and slightly depends on k. Figure 3.5a illustrates the effect of n_s on precomputation time. The precomputation time increases from 82 seconds to 407 seconds, when n_s increases from 5 million to 25 million. As evident in Figure 3.5b, the effect of k is not sharp as n_s. For instance, the precomputation takes 82 seconds when k is 100. When k becomes



(a) Precomputation Time vs. n_s



(b) Precomputation Time vs. k



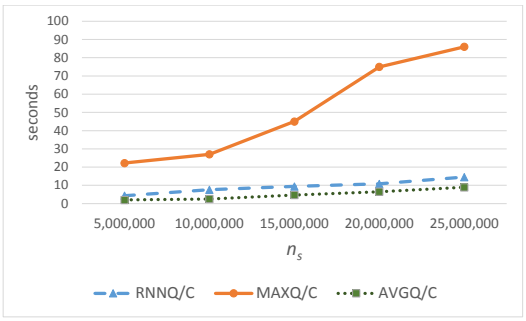
(c) Precomputation Time vs. *maxCoordinate*

Figure 3.5: Precomputation times of the client-based protocols.

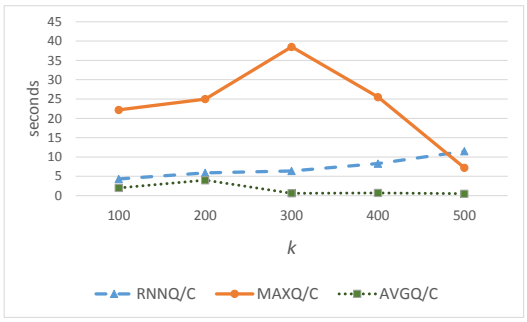
500, the time increases to 110 seconds. For the NYC and Tokyo datasets, the precomputation time is 4 seconds and 9.6 seconds, respectively, due to the lower n_s values in these datasets.

The query processing time of RNNQ/C depends on the values of n_s and k. Although an increase in k decreases the workload of the server, it increases the total number of decryptions performed by the client. Figure 3.6a and Figure 3.6b shows the query processing time for different values of n_s and k. These two variables are not the only factors that determine the query processing time because the total number of operations also depends on the total number of users attracted by the new facility. Query processing takes 1.9 seconds and 2.5 seconds, for the NYC and Tokyo datasets, respectively.

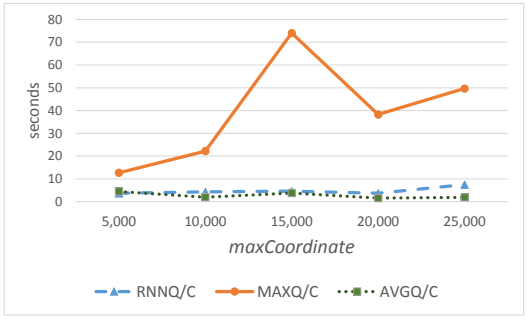
We also evaluate the amortized computation time of RNNQ/C for 100 queries. For the parameters given above, the precomputation takes 82 seconds and the query processing takes 4.3 seconds. Hence, the amortized computation time per



(a) Query Processing Time vs. n_s



(b) Query Processing Time vs. k



(c) Query Processing Time vs. *maxCoordinate*

Figure 3.6: Query processing times of the client-based protocols.

query is 5.1 seconds.

During query processing, k ciphertexts and the facility locations are shared between the server and the client. Hence, k is the most crucial parameter for the communication cost. When the total number of facilities is 100, the amount of shared data is nearly 25 KB.

AVGQ/C. In this protocol, the client obtains the total number of users in $\mathcal{U}_{\mathcal{I}}$ and the total distance between each user and her nearest facility. Until there is a change on the total number of users, there is no need to compute it in each query. Hence, the total number of users in $\mathcal{U}_{\mathcal{I}}$ can be precomputed by the server and the client. This part of precomputation requires n_s multiplications, one encryption, and one decryption. In addition, the server can precompute $n_s \cdot k$ distance calculations, n_s exponentiations, n_s multiplications, and one encryption for the computation of the total distance. During query processing the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, nearly $\frac{n_s}{k}$ exponentiations,

and nearly $\frac{n_s}{k}$ modular inverse calculations. The client performs one decryption and one division during query processing.

Due to the high number of exponentiations, the precomputation time of AVGQ/C is higher than the other client-based protocols. Figure 3.5a depicts the precomputation time for different values of n_s. Query processing takes nearly 12 minutes when the server has 5 million users and the time changes linearly with respect to the value of n_s. For the smaller n_s values in NYC and Tokyo datasets, the precomputation takes 41 seconds and 92 seconds, respectively. In addition, Figure 3.5c shows that the value of $maxCoordinate$ affects the precomputation time slightly. As $maxCoordinate$ increases, exponent values in the computation also increase.

The query processing time of AVGQ/C is directly proportional to n_s and inversely proportional to k. Figure 3.6a and Figure 3.6b shows the query processing time for different values of n_s and k. Since there is no encryption and only one decryption in query processing, AVGQ/C has the lowest query processing time among three client-based protocols. Query processing takes nearly 1 second, for both NYC and Tokyo datasets. Similar to RNNQ/C, the total number of users attracted by the new facility affects the query processing time. The precomputation takes 720 seconds and the query processing takes 2 seconds for the parameters given above. Hence, the amortized computation time per query is 9.2 seconds for 100 AVGQ/C queries.

During the protocol, two ciphertexts and the facility locations are shared between the server and the client. Therefore, the communication cost is low because the facility locations are sent as plaintext and the total number of ciphertexts is two. When the total number of facilities is 100, the amount of shared data is less than 1 KB during query processing.

MAXQ/C. In MAXQ/C, the server can precompute $n_s \cdot k$ distance calculations, n_s multiplications and w encryptions. During query processing, the server performs n_s distance calculations, nearly $2 \cdot \frac{n_s}{k}$ multiplications, nearly $\frac{n_s}{k}$ modular inverse calculations, and w encryptions. The client also performs $w - q + 1$

decryptions during query processing.

w and n_s are crucial parameters in the precomputation time of MAXQ/C. Similar to RNNQ/C protocol, the precomputation time mostly depends on n_s because n_s is much higher than w in our experiments. Figure 3.5a shows the precomputation time with respect to n_s. Since n_s multiplications dominate the computation cost, the precomputation time of MAXQ/C is similar to RNNQ/C. We also observed similar results for the real datasets.

Due to the randomness in the selection of w, the query processing time of MAXQ/C is not directly proportional to n_s or $maxCoordinate$. w is a randomly selected value that is greater than max, which is the maximum distance between a user in $\mathcal{U}_\mathcal{S}$ and her nearest facility. In our experiments, w is selected randomly in the range $[max,\ 2 \cdot max]$. Therefore, the parameter $maxCoordinate$ affects the value of w, and hence the query processing time. The query processing time of MAXQ/C for different values of $maxCoordinate$ is given in Figure 3.6c. As evident in Figure 3.6c, query processing time is not directly proportional to $maxCoordinate$. For instance, when $maxCoordinate$ increases from 15,000 to 20,000, the computation time decreases due to a decrease in w and an increase in q. Therefore, the distance of each user to her nearest facility and the query result q also affect the query processing time. In addition, Figure 3.6b shows the query processing time for different values of k. Smaller k values may result in higher query processing times because max value may increase in smaller k values. For instance, in real datasets we have smaller k values such as 10 and 20. As a result, the query processing times are 17 seconds and 60 seconds, for the NYC and Tokyo datasets, respectively.

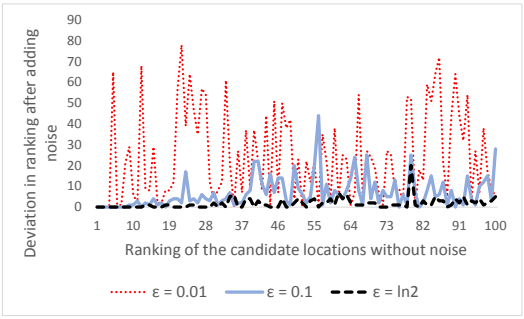
For the parameters given above, the precomputation takes 83 seconds and the query processing takes 22.2 seconds. Therefore, the amortized computation time per query is 23 seconds for 100 MAXQ/C queries. The communication cost of the protocol is w ciphertexts and the facility locations. When w is selected as 5000, the amount of shared data is nearly 1.25 MB.

All these results show the practicality of our proposed scheme in real-life settings. Any of the aforementioned queries can be performed in less than a minute on datasets that include millions of individuals.

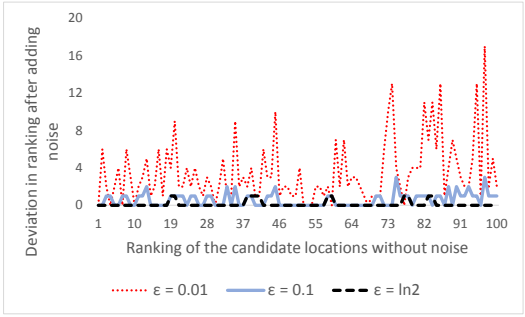
3.5.3 Utility vs. Differential Privacy

In Section 3.4, we explain how to achieve differential privacy in the proposed protocols by adding controlled noise to the query results, which affects the accuracy of the results. To measure the utility of the protocols under differential privacy, we selected 100 candidate locations for the new facility and observed the results after executing the protocols. We divided the whole region into a 10x10 grid and selected the center of each grid as a candidate location for the new facility. First, we applied the protocols without adding any noise and ranked the 100 candidate locations with respect to their optimality. Then, we executed the protocols by adding controlled noise and observed the impact of differential privacy on the utility. We evaluated the utility of differential privacy for the real and synthetic datasets. For synthetic datasets, we set the parameters as given in Section 3.5.2.

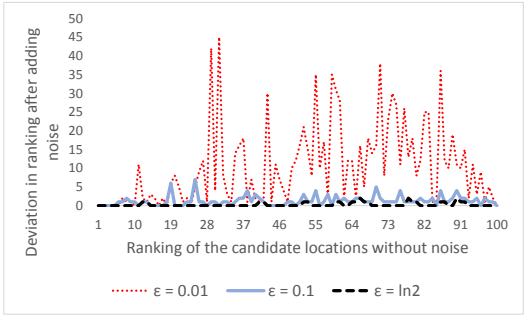
RNNQ. The objective of using RNNQ is uniformly distributing the cardinality of the RNNs. When the new facility attracts users from dense facilities, the workloads of dense facilities decrease. Hence, balancing workload reduces the wait times by avoiding overloads. We measured the standard deviation of the cardinalities of the RNNs. We sorted 100 candidate locations with respect to the standard deviation after adding the possible location as the new facility. The best candidate is the location that minimizes the standard deviation. We also sorted the candidate locations after achieving differential privacy. Figure 3.7 shows the rankings of the candidates for real and synthetic datasets after adding controlled noise. We used three different ϵ values such as 0.01, 0.1, and $\ln 2$, which are typically chosen values in the literature. As evident in Figure 3.7, the utility increases when ϵ increases. When ϵ is $\ln 2$, the ranking of the candidates are almost same as the rankings without adding any noise. Although the deviations in the rankings increase for the smaller values of ϵ, the best candidate is same



(a) NYC Dataset

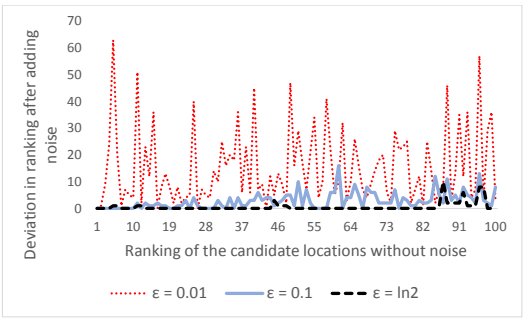


(b) Tokyo Dataset

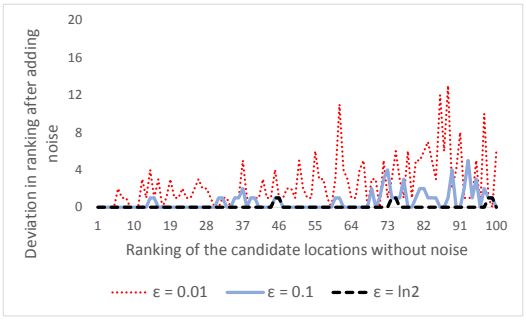


(c) Synthetic Dataset

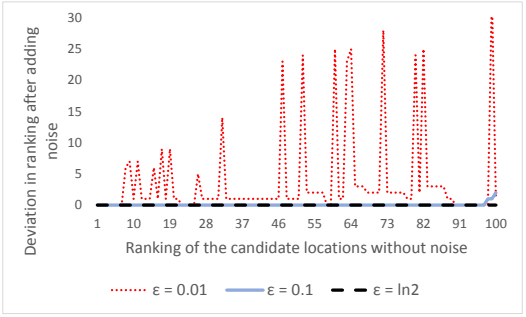
Figure 3.7: Deviation in the rankings of the 100 candidate locations after achieving differential privacy in RNNQ. x axis represents the ranking of the candidate locations when RNNQ is performed with exact query results. y axis represents the change in the ranking of each candidate location when differential privacy is achieved in RNNQ. For instance, in Figure 3.7a the point $(5, 65)$ for $\epsilon = 0.01$ shows that the 5th best candidate location for the new facility becomes 70th best location after adding noise to the query result. Depending on the maximum change in the ranking, the range of y axis varies for each dataset.



(a) NYC Dataset



(b) Tokyo Dataset



(c) Synthetic Dataset

Figure 3.8: Deviation in the rankings of the 100 candidate locations after achieving differential privacy in AVGQ.

most of the time after achieving differential privacy. Therefore, the utility of RNNQ under differential privacy is remarkable for large ϵ values and acceptable for small ϵ values.

AVGQ. We sorted 100 candidate locations with respect to the average distance value returned from the query. The best candidate is the location that minimizes the average distance. Figure 3.8 shows the rankings of the candidates after adding controlled noise. The deviation on the rankings is less than RNNQ. Therefore, the utility of AVGQ under differential privacy is better than RNNQ for all ϵ values.

MAXQ. This query returns w values containing zero and non-zero elements. The largest index of a non-zero element is the result of the query. After adding the noise to each of w values, most of the zero values becomes non-zero. Therefore, adding the noise changes the query result significantly. In our experiments, we

observed that the query result becomes w or $w-1$ most of the time after adding the noise. Since w is a randomly selected value, the query returns a random result in each execution. Hence, the utility of MAXQ under differential privacy is very low.

Discussion. Our experimental results show that achieving differential privacy in RNNQ and AVGQ causes low utility loss. Since these queries contain counting subqueries, running them under differential privacy increases the privacy of individuals with a negligible computational overhead. On the other hand, MAXQ is not suitable for differential privacy because the query result changes significantly after adding noise to each counter value in the query. To prevent high utility loss of differential privacy in MAXQ, only non-zero values should be randomized as described in Section 3.3.3.

3.6 Conclusion

In this chapter, we proposed protocols for privacy-preserving analysis of location data in a location-based service provider (referred as the server) by a business (referred as the client) as a service. We defined three queries addressing different objectives in optimal location selection: (i) to minimize the average distance between each user and her closest facility, (ii) to minimize the maximum distance between a user and her closest facility, and (iii) to uniformly distribute the workload in facilities. We developed two homomorphic encryption-based solutions: (i) a server-based solution, in which most of the computation is performed by the server, and hence the workload of the client is low, and (ii) a client-based solution, in which the client performs the majority of the computation during the setup phase (which only occurs once) and after completion of the setup phase, all queries are processed quickly. We showed that the proposed protocols keep the client's user list and the query result hidden from the server, and the location information stored at the server hidden from the client. The security provided by all protocols relies on the underlying security of the Paillier cryptosystem (which relies on the decisional composite residuosity assumption) proved in [30]. We

also showed that it is possible to achieve differential privacy in the proposed protocols with low utility loss. Using the proposed protocols will facilitate sharing location information between entities without compromising customer privacy. We evaluated the efficiencies of the proposed protocols through experiments for each considered query type and showed that the proposed protocols are feasible, efficient, and scalable.

Chapter 4

Synthetic Location Data Generation

In this chapter, we focus on generating synthetic data and using it in optimal location queries. In Section 4.1, we present an optimal location predictor that works at the client side by generating synthetic data when the client has partial information about user locations. We also propose two differentially private data generation methods in Section 4.2 which can be used for private data publishing by the server. In Section 4.3, we give the experimental results for the data generation methods.

4.1 Data Generation at Client Side

In this section, we study the following setting for the optimal location problems: The client knows some partial information about user locations such as the total number of users attracted by each existing facility and it wants to find the optimal location using this partial information without interacting with the server. Since there is no communication with the server, we refer the client as the business. We propose an optimal location predictor [7] which accepts partial information

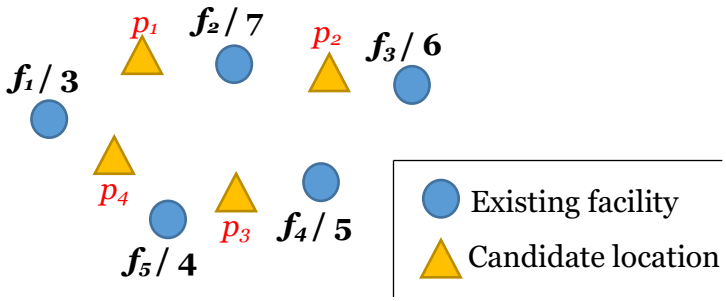


Figure 4.1: An example scenario for the problem when partial information is known by the business.

about user locations and returns a location for the new facility.

Although many businesses do not know exact locations of their customers, they naturally have the number of customers for each existing facility. The density of customers may also be known as a result of privacy-preserving RNN Cardinality Query defined in Section 3.1.2.1. Figure 4.1 shows an example scenario for the addressed problem, where a business has five existing facilities and it has the knowledge of total number of users attracted by each facility. For instance, there are 3 users whose nearest facility is f_1 in Figure 4.1. The business needs to decide the best location among the candidates to open a new facility.

We develop a method that generates synthetic user locations by using the partial information known by the business and predicts the optimal location after running the query on generated location data. User locations are generated based on the total number of users attracted by each facility. We form the Voronoi diagram for existing facilities and generate the users in each facility in its Voronoi region. Instead of just uniformly distributing the users within each Voronoi region, we use the density of users in neighbor facilities in Voronoi diagram by dividing the Voronoi region of each facility into triangular regions and generating the user locations in these smaller regions. It is possible to apply other constraints in the generation of user location data such as removing the areas where no one lives (e.g. seas and forests). We performed experiments on real datasets containing Foursquare check-ins in New York City and Tokyo to show how each additional information increases the accuracy of the predictor.

Table 4.1: Notations used in Section 4.1.

$\mathcal{F}$	the set of existing facilities
$\mathcal{U}$	the set of users
$\mathcal{P}$	the set of candidate locations for the new facility
$d(a, b)$	distance between points a and b
$\mathcal{I}(p_i)$	the set of users attracted by the new facility if it is built at location p_i
$BRNN(f_i)$	the set of users attracted by the facility f_i for the given set of existing facilities
$\mathcal{A}(p_i)$	the average distance from each user to her nearest facility if the new facility is built at location p_i
$\|\mathcal{S}\|$	the cardinality of set $\mathcal{S}$
x_p	abscissa of the point p in Euclidean space
y_p	ordinate of the point p in Euclidean space
$\mathcal{R}$	the region considered by the generator
$\mathcal{R}_i$	Voronoi region of the facility f_i
$\mathcal{R}_{i,j}$	triangular region in $\mathcal{R}_i$

4.1.1 Problem Formulation

We first define max-inf and min-dist optimal location queries and then list the partial and auxiliary information that may be known by businesses to run these queries. Table 4.1 summarizes the notations used in this chapter. All the data objects are represented by points in Euclidean space.

Definition 4 *Given a set $\mathcal{F}$ of existing facilities, a set $\mathcal{U}$ of users, and a set $\mathcal{P}$ of candidate locations, the* ***max-inf optimal location query*** *finds a location $p \in \mathcal{P}$ for a new facility such that $\forall p' \in \mathcal{P}$,*

$$|\mathcal{I}(p)| \geq |\mathcal{I}(p')|$$

where $\mathcal{I}(p_i) = \{u \mid u \in \mathcal{U} \wedge \forall f \in \mathcal{F}, d(u, p_i) \leq d(u, f)\}$.

Definition 5 *Given a set $\mathcal{F}$ of existing facilities, a set $\mathcal{U}$ of users, and a set $\mathcal{P}$ of candidate locations, the* ***min-dist optimal location query*** *finds a location $p \in \mathcal{P}$ for a new facility such that $\forall p' \in \mathcal{P}$,*

$$\mathcal{A}(p) \leq \mathcal{A}(p')$$

where $\mathcal{A}(p_i) = \frac{\sum_{u \in \mathcal{U}} \{d(u,f_i) \mid f_i \in \mathcal{F} \cup p_i \ \wedge \ \forall f_j \in \mathcal{F} \cup p_i, \ d(u,f_i) \leq d(u,f_j)\}}{|\mathcal{U}|}$.

The above definitions of optimal location queries state that the set $\mathcal{U}$ must be provided. However, in our problem setting, the business that wants to run optimal location queries does not own the set of user locations ($\mathcal{U}$). We assume that the business knows the total number of users attracted by each facility. Formally, for each facility $f \in \mathcal{F}$, $|BRNN(f)|$ is known by the business where $BRNN(f_i) = \{u \mid u \in \mathcal{U} \wedge \forall f_j \in \mathcal{F}, d(u, f_i) \leq d(u, f_j)\}$. To run optimal location queries, the business can generate a set $\mathcal{U}'$ to mimic $\mathcal{U}$ based on the total number of users attracted by each facility. However, businesses may have more yet partial information about user locations. Here, we list auxiliary information (AI) that may be known by businesses and we explain how to use such partial information during the generation of user locations in Section 4.1.2.

AI 1 *The business may know the overall* ***minimum and maximum values*** *for x and y coordinates in Euclidean space. These values can be represented as follows:*

- $x_{min} = min\{x_u \mid u \in \mathcal{U}\}$
- $y_{min} = min\{y_u \mid u \in \mathcal{U}\}$
- $x_{max} = max\{x_u \mid u \in \mathcal{U}\}$
- $y_{max} = max\{y_u \mid u \in \mathcal{U}\}$

AI 1 provides the minimum bounding rectangular region for the user locations. Figure 4.2 shows an example in which the black and orange points represent users. AI 1 indicates that all user data are inside the green rectangle for the example in Figure 4.2.

AI 2 *The business may know the* ***minimum bounding convex polygon*** *of the user locations.*

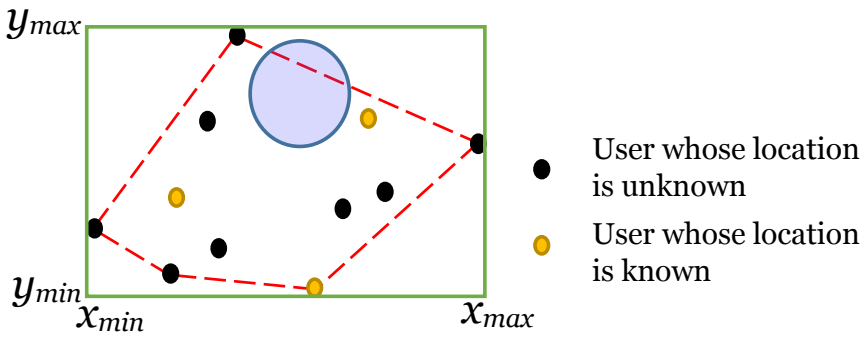


Figure 4.2: An example scenario for auxiliary information that may be known by a business.

AI 2 provides the convex hull for the user locations. In Figure 4.2, the user locations are bounded with a pentagon drawn with red dotted lines. Hence, the data generator should generate the all users inside this polygon if AI 2 is known.

AI 3 *The business may know* ***empty regions*** *which do not contain any user.*

The business can avoid generating synthetic user data in regions where no one lives (e.g. seas and forests). For instance, blue circle in Figure 4.2 represents a lake. Therefore, the data generator should not generate a user location inside this region.

AI 4 *The business may know a* ***subset of*** $\mathcal{U}$*.*

Although the business does not know $\mathcal{U}$ in the problem setting, locations of some users may be known. In Figure 4.2, orange points represent the users whose locations are known by the business. Therefore, during data generation it is enough to generate the locations for the other users, who are represented with black points. In Section 4.1.2, we present the proposed predictor and explain the usage of auxiliary information during data generation. We also analyze the effect of each one on the accuracy of the predictor in Section 4.3.

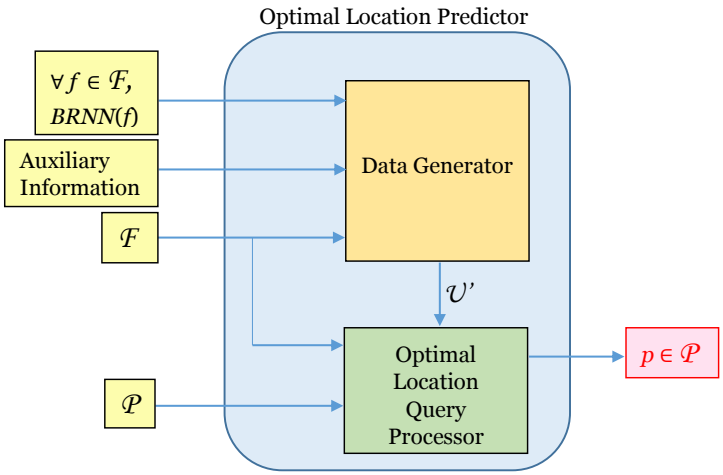


Figure 4.3: Optimal Location Predictor.

4.1.2 Predicting Optimal Location

In this section, we present our optimal location prediction mechanism, when the business knows only $|BRNN(f)|$ for each facility $f \in \mathcal{F}$. The business may also know auxiliary information about user locations. To run optimal location queries, $\mathcal{F}$, $\mathcal{U}$, and $\mathcal{P}$ must be given. Since the business does not own the set $\mathcal{U}$, we propose a location data generator to produce synthetic user locations $\mathcal{U}'$ that mimics $\mathcal{U}$. The query processor then returns the optimal location p for given $\mathcal{F}$, $\mathcal{U}'$, and $\mathcal{P}$. Figure 4.3 shows how our predictor works.

Along with the $|BRNN(f)|$ for each facility $f \in \mathcal{F}$, the data generator needs a region $\mathcal{R}$ for generating users in this region. If the business knows AI 1, the data generator uses the minimum bounding rectangle as $\mathcal{R}$. Otherwise, the business selects a region $\mathcal{R}$ that will include all synthetic user locations. To represent $\mathcal{R}$ in figures clearly, we used a rectangular region. However, it is not necessary to use a rectangular region. In this region $\mathcal{R}$, the generator locates the existing facilities ($\mathcal{F}$) and creates the Voronoi diagram which is a partitioning of a plane into convex polygons such that each polygon contains one existing facility $f_i \in \mathcal{F}$. Voronoi region of each facility $f_i \in \mathcal{F}$ is the set of all points in $\mathcal{R}$ whose distance to f_i is not greater than their distance to the other facilities. Formally, the Voronoi

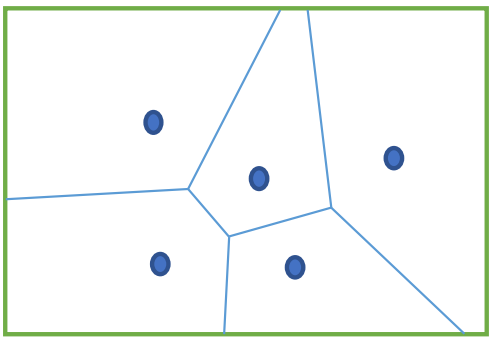


Figure 4.4: An example region $\mathcal{R}$ after Voronoi Diagram is created by data generator.

region of facility $f_i \in \mathcal{F}$ is

$$\mathcal{R}_i = \{r \in \mathcal{R} \mid \forall f_j \in \mathcal{F},\ d(r, f_i) \leq d(r, f_j)\}$$

An example Voronoi diagram for 5 facilities can be seen in Figure 4.4. After creating the Voronoi diagram, the generator identifies the regions in $\mathcal{R}$ which do not contain any user by checking AI 2 and AI 3. If AI 2 is provided, the generator eliminates the regions in $\mathcal{R}$ but not in the minimum bounding polygon during data generation. If some other empty regions which do not contain a user (i.e. AI 3) are provided, the generator also eliminates these regions. In Figure 4.5, these eliminated regions are represented with black. For AI 3, the generator accepts empty regions as polygons. Hence, the business enters the coordinates of the vertices of the polygons for AI 2 and AI 3. In addition, if the business knows a subset of $\mathcal{U}$ (i.e. AI 4), the locations of these users are inserted into $\mathcal{R}$. In Figure 4.5, orange points represent the users whose locations are known by the business. Therefore, they will be included in $\mathcal{U}'$.

After considering auxiliary information, the data generator starts generating user locations for each facility $f_i \in \mathcal{F}$. For a facility f_i, the generator needs to generate $|BRNN(f_i)|$ users in its Voronoi region $\mathcal{R}_i$. $\mathcal{R}_i$ is a convex polygon and each edge of the polygon is either a common edge with a neighbor facility or a segment of an edge of $\mathcal{R}$. It is expected that there are more users in the subregions of $\mathcal{R}_i$ which are close to neighbor facilities with high density of users. Hence, rather than assigning these points uniformly random in each Voronoi region, we

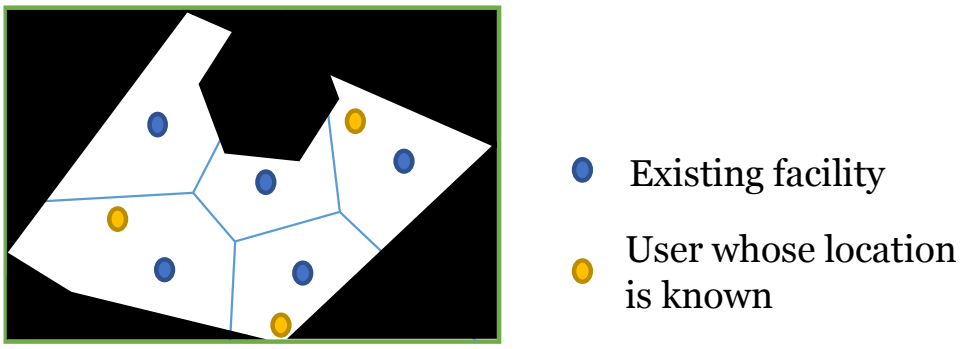


Figure 4.5: An example region $\mathcal{R}$ after auxiliary information is considered by data generator.

use the number of users attracted by each neighbor facility in Voronoi diagram by assigning a weight. To use the density of users in neighbors, we divide the region of each facility into triangular regions by connecting each facility $f_i \in \mathcal{F}$ with the vertices of its Voronoi region $\mathcal{R}_i$. For the given example in Figure 4.6, $\mathcal{R}_i$ is divided into 5 triangular regions. The data generator decides the total number of users to be generated in each triangular region based on:

1. the area of the region,
2. the total number of users attracted by its neighbor facility.

Let the total number of triangular regions in $\mathcal{R}_i$ be m_i and these regions be $\{\mathcal{R}_{i,1}, ..., \mathcal{R}_{i,m_i}\}$. For a region $\mathcal{R}_{i,j}$, let $BRNN$ of its neighbor be $n_{i,j}$. Let $n_i = \sum_{k=1}^{m_i} n_{i,k}$. Hence, n_i is the total number of users attracted by all of the neighbors of f_i. Then, the total number of users to be generated in a triangular region $\mathcal{R}_{i,j}$ is calculated as

$$|BRNN(f_i)| \cdot \left(\omega \cdot \frac{n_{i,j}}{n_i} + (1 - \omega) \cdot \frac{Area(\mathcal{R}_{i,j})}{Area(\mathcal{R}_i)} \right)$$

In this formula, ω is the weighting factor that represents the effect of $n_{i,j}$ on the total number of users to be generated in $\mathcal{R}_{i,j}$. When ω is selected as 0, the generator distributes users with respect to the area of each triangle in $\mathcal{R}_i$ without considering the number of users attracted by neighbors.

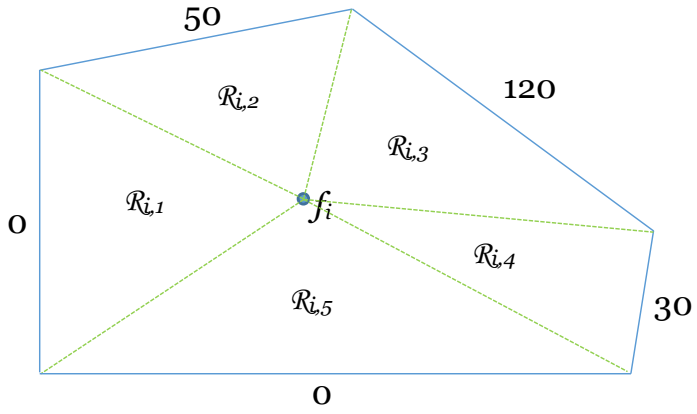


Figure 4.6: Dividing $\mathcal{R}_i$ into triangular regions.

For instance, if $|BRNN(f_i)|$ is 50 and $\frac{Area(\mathcal{R}_{i,3})}{Area(\mathcal{R}_i)}$ is $\frac{1}{5}$ in Figure 4.6, the generator generates $50 \cdot \left(0.5 \cdot \frac{120}{200} + (1 - 0.5) \cdot \frac{1}{5}\right) = 20$ users in $\mathcal{R}_{i,3}$ if ω is selected as 0.5. For different values of ω in the range of $[0, 1]$, the total number of users to be distributed in $\mathcal{R}_{i,3}$ varies between 10 and 30.

By using the given formula, the generator decides the number of users in each triangular region and produces the user locations. To produce a random location inside a triangle, one can select three random points s_1, s_2, s_3 in the range of [0,1] such that $s_1 + s_2 + s_3 = 1$ and use these three points as barycentric coordinates of the random point inside the triangle. For a triangle with vertices P_1, P_2, and P_3, the random point can be determined as $s_1 \cdot P_1 + s_2 \cdot P_2 + s_3 \cdot P_3$.

If the locations of some users are given as auxiliary information (i.e. AI 4), the generator generates the locations for the other users. If some part of the triangular region is removed by AI 2 or AI 3, the area of the remaining region is considered in the formula.

After generating synthetic user locations, optimal location query is executed by the predictor. In max-inf optimal location query, the size of the influence set ($|\mathcal{I}(p_i)|$) for each candidate $p_i \in \mathcal{P}$ is calculated. The candidates are ranked with respect to sizes of their influence sets and the candidate with maximum size is returned as the best candidate. In min-dist optimal location query, the average

distance ($\mathcal{A}(p_i)$) from each user to nearest facility is calculated if the new facility is built at the location p_i. Similarly, the candidates are ranked with respect to the average distance values and the candidate with minimum value is returned as the best candidate.

4.2 Data Generation at Server Side

Synthetic data generation is an alternative for data anonymization. In our setting, the server can generate synthetic location data that keeps the characteristics of the original data and share it with the client. The client can use synthetic data in optimal location selection as explained in Section 4.1. Synthetic data also allows a wide variety of analytics tasks besides optimal location queries.

Privacy-preserving spatial data publication has been extensively studied in the literature as summarized in Section 2.6. The general approach is to generate private histograms over multi-dimensional partitions. A query processing component is also usually provided to maximize the utility for a given type of query, such as range query, over the released information.

While a workload aware statistics can provide theoretical guarantees for the given queries, the format of such statistics is usually different from the original data set. Moreover, such statistics usually come with a budget of differential privacy. Releasing a synthetic data set in the same format as the original data gives more flexibility in how the clients use it with no concerns of query based of privacy budgets. In many practical scenarios, the clients want to be able to utilize their in-house data analytical tools in full force and do not want any additional requirements in how they analyze the data. They may define new queries which may not be known at the time of the release.

Here, we present a method to generate synthetic data sets that support all analytics tasks that the original data can be used to answer. For privacy-preserving synthetic data generation, Lu et al. [47] propose to define the model of the

database by a set of counting queries. The data owner adds noise to the result of counting queries and generates synthetic data based on the noisy results. Therefore, generated data satisfies differential privacy. To apply this approach into location data, we propose a simple method by dividing the whole region into a grid. In addition, we propose a clustering-based data generation method to keep the characteristics of the original data better.

4.2.1 Grid-Based Data Generation

In this method, the server divides the whole region into a grid and counts the number of users in each grid cell. Then, the server adds noise to the number of users in each grid cell to satisfy differential privacy. The server determines the amount of noise as explained in Section 3.4. Since counting queries in each grid cell can be thought as a histogram query, the sensitivity is 2. Hence, the server adds a noise $Laplace(\frac{2}{\epsilon})$ to the number of users in each grid cell to satisfy differential privacy.

As explained in [46], the accuracy of query results can deteriorate in the grid-based method when there is a large number of cells, due to low counts in grid cells since the output contains large mass of noisy counts. Hence, we guarantee generating at least k users in each grid cell by merging the cells having less than k users. Having k users in each region also prevents linking of real database items with the synthetic ones. Otherwise, an adversary, who knows that a real user is located in a grid cell, can link that user with a synthetic one with a probability more than $1/k$ if there is less than k synthetic users in that grid cell. The steps of the method are as follows:

- **Step 1:** Divide the data set into a grid that consists of m subregions (i.e. grid cells) such as $\mathcal{R} = \{R_1, ..., R_m\}$.
- **Step 2:** Count the number of users n_i in each subregion R_i.
- **Step 3:** Add a noise $Laplace(\frac{2}{\epsilon})$ to n_i of each subregion and calculate

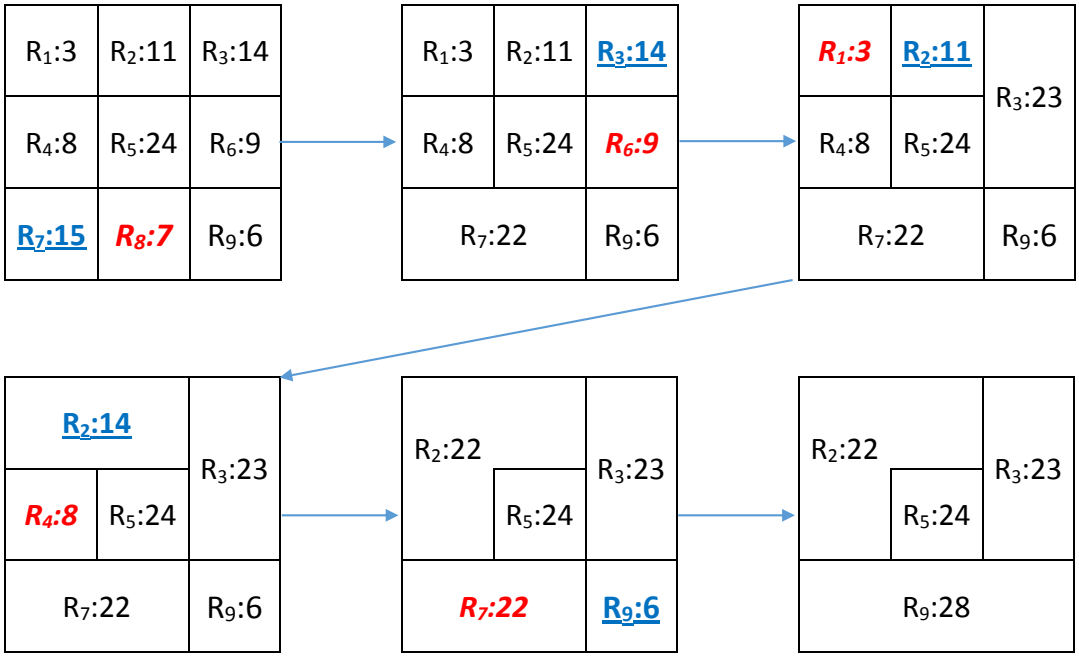


Figure 4.7: An example execution of grid-based data generation algorithm.

$n'_i = n_i + Laplace(\frac{2}{\epsilon})$. Hence, $\mathcal{N}' = \{n'_1, ..., n'_m\}$ is the set that keeps the number of users to be generated in each subregion.

- **Step 4:** Determine the neighbors G_i of each subregion R_i. Then, $\mathcal{G} = \{G_1, ..., G_m\}$ is the neighbors of all subregions.
- **Step 5:** Determine the subregion $R_i \in \mathcal{R}$ with the maximum n' value less than k. For such R_i, find its neighbor $R_j \in G_i$ with the minimum n' value by checking its neighbors.
- **Step 6:** Update region R_i by merging it with R_j. Update n'_i by adding n'_j. Update the neighbors G_i with the union of G_i and G_j. Change all R_j values in $\mathcal{G}$ with R_i. Remove R_j, n'_j, and G_j from $\mathcal{R}$, $\mathcal{N}'$, and $\mathcal{G}$, respectively.
- **Step 7:** Repeat Steps 5 and 6 iteratively until n' of each region is greater than or equal to k.
- **Step 8:** Generate n'_i synthetic users in each subregion $R_i \in \mathcal{R}$.

Figure 4.7 shows an example for the execution of the algorithm. The first grid

shows all subregions ($\mathcal{R}$) and the number of users in each subregion after adding noise ($\mathcal{N}'$). For example, there are 3 users to be generated in the subregion R_1. Let the parameter k be 20 in the example. Hence, the subregions other than R_5 must be merged with the neighbors. In each step, the algorithm finds the subregion with highest n' less than 20 and merges it with a neighbor. At the end, synthetic data is generated in 4 regions, each of which has at least 20 users.

4.2.2 Clustering-Based Data Generation

We propose a data generation method based on clustering which aims to preserve the characteristics of the original data. Clustering is grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other. There are several types of clustering algorithms such as connectivity-based clustering, centroid-based clustering, distribution-based clustering, and density-based clustering.

We use K-means clustering algorithm which is a popular centroid-based clustering technique. The goal of K-means clustering is dividing data into K groups each of which are represented by a centroid. In K-means, K points are selected randomly as cluster centroids initially. Each object is assigned to the closest centroid and the new centroid of each cluster is calculated based on the assignments. Then, objects are assigned to the new centroids and the process continues until the same objects are assigned to each cluster in consecutive rounds. Since the number of clusters (i.e. K) is decided randomly before the algorithm, the number of objects in each cluster can be variable.

In this algorithm, our aim is to have at least k users in each cluster in order to achieve balanced clustering and prevent linking of users. To have at least k users in each cluster, similar to the grid-based data generation, clusters with less than k users are merged with the neighbors. However, unlike grid-based clustering, the areas of subregions are not fixed at the beginning of the algorithm. Subregions of some clusters can be small and violate privacy. We use a threshold θ for minimum subregion size. Hence, as the last step of the algorithm, clusters whose

area are less than θ are merged with the neighbors. At the end of the algorithm, all clusters must have at least k users and their areas must be at least θ.

In the algorithm, we first run K-means algorithm with a large K value to obtain the initial clusters. Similar to the grid-based algorithm, the server adds a noise $Laplace(\frac{2}{\epsilon})$ to the number of users in each cluster to satisfy differential privacy. Then, merging step guarantees having at least k users in each cluster. In K-means clustering, forming a Voronoi diagram based on cluster centroids provides a partitioning of the whole region into Voronoi regions of each cluster. Therefore, after finding the desired clustering, data generation is performed in each Voronoi region.

Instead of generating synthetic users uniformly in the regions, some properties of the cluster can be preserved such as the noisy total distance from the cluster centroid to all users in the cluster. To satisfy differential privacy, controlled random noise must be added to the total distance calculated in each cluster. As explained in Section 3.4, for each cluster the sensitivity is the maximum distance (max) between a user in the cluster and the cluster centroid. Hence, the server adds a noise $Laplace(\frac{max}{\epsilon})$ to the total distance to satisfy differential privacy. Noisy total distances for each cluster are preserved in the final synthetic data. The steps of our clustering-based data generation algorithm are as follows:

- **Step 1:** Apply K-means clustering for a large value of K. Let $\mathcal{T} = \{T_1, ..., T_K\}$ be the cluster centroids, $\mathcal{R} = \{R_1, ..., R_K\}$ be the Voronoi regions of each cluster.

- **Step 2:** Count the number of users n_i in each Voronoi region R_i. Add a noise $Laplace(\frac{2}{\epsilon})$ to n_i of each Voronoi region and calculate $n'_i = n_i + Laplace(\frac{2}{\epsilon})$. Hence, $\mathcal{N}' = \{n'_1, ..., n'_K\}$ is the set that keeps the number of users to be generated in each Voronoi region.

- **Step 3:** Calculate the total distance d_i from cluster centroid T_i to all users in the cluster. Let $U_i = \{u_{i,1}, ...u_{i,n_i}\}$ be the real users in region R_i. Calculate the total distance $d_i = \sum_{i=1}^{n_i} dist(T_i, u_i)$. Let the maximum distance of

the cluster centroid T_i to any user in region R_i be max_i. Add $Laplace(\frac{max_i}{\epsilon})$ to d_i and calculate $d'_i = d_i + Laplace(\frac{max_i}{\epsilon})$. Hence, $\mathcal{D}' = \{d'_1, ..., d'_K\}$ is the noisy total distances for each region that should be preserved in synthetic data generation.

- **Step 4:** Determine the neighbors G_i of each Voronoi region R_i. Then, $\mathcal{G} = \{G_1, ..., G_m\}$ is the neighbors of all Voronoi regions.

- **Step 5:** Determine the Voronoi region $R_i \in \mathcal{R}$ with the maximum n' value less than k. For such R_i, find its neighbor $R_j \in G_i$ with the minimum n' value by checking its neighbors.

- **Step 6:** Update region R_i by merging it with R_j. Update n'_i by adding n'_j. Update d'_i by adding d'_j. Update the neighbors G_i with the union of G_i and G_j. Change all R_j values in $\mathcal{G}$ with R_i. Remove R_j, n'_j, d'_j, and G_j from $\mathcal{R}$, $\mathcal{N}'$, $\mathcal{D}'$, and $\mathcal{G}$, respectively.

- **Step 7:** Repeat Steps 5 and 6 iteratively until n' of all regions is greater than or equal to k.

- **Step 8:** Determine the Voronoi region $R_i \in \mathcal{R}$ with the largest area smaller than the threshold θ. For such R_i, find its neighbor $R_j \in G_i$ with the smallest area by checking its neighbors.

- **Step 9:** Update region R_i by merging it with R_j. Update n'_i by adding n'_j. Update d'_i by adding d'_j. Update the neighbors G_i with the union of G_i and G_j. Change all R_j values in $\mathcal{G}$ with R_i. Remove R_j, n'_j, d'_j, and G_j from $\mathcal{R}$, $\mathcal{N}'$, $\mathcal{D}'$, and $\mathcal{G}$, respectively.

- **Step 10:** Repeat Steps 8 and 9 iteratively until the area of each region is greater than or equal to θ.

- **Step 11:** Generate n'_i synthetic users in each Voronoi region $R_i \in \mathcal{R}$. Initially set $d''_i = 0$. To keep the total distance d'_i in data generation, the distance of each generated user in R_i to the cluster centroid is added to d''_i. In each generated user, compare the average distance of synthetic users to the cluster centroid with d'_i/n'_i. If the average distance is greater than d'_i/n'_i, reduce the size of R_i for next synthetic users.

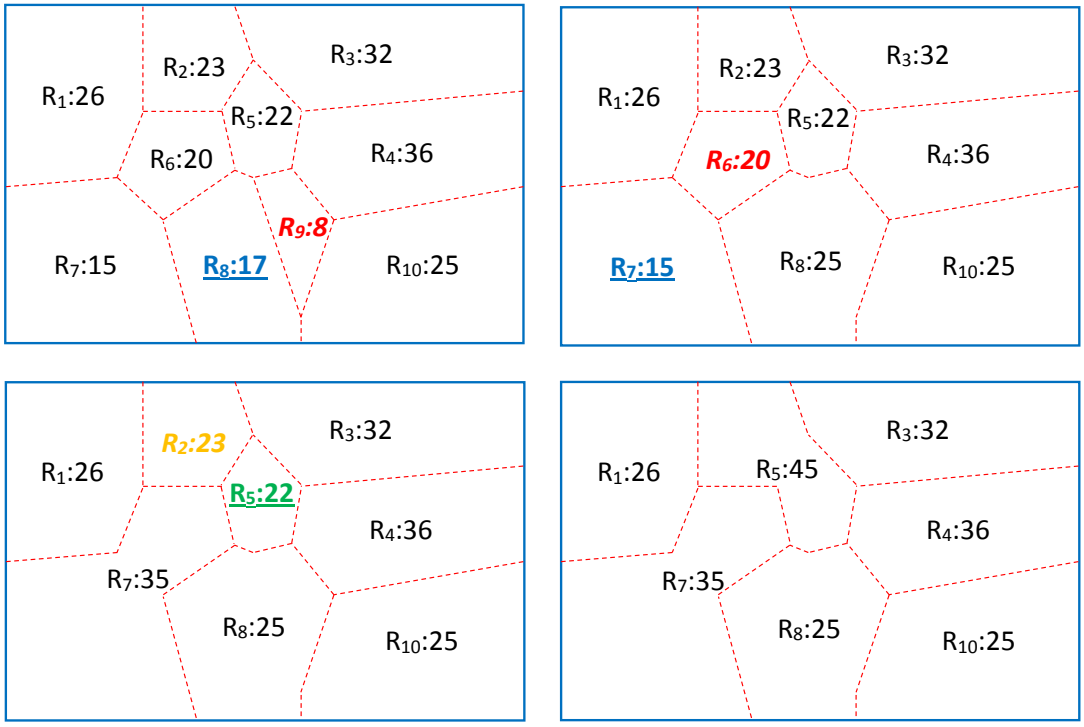
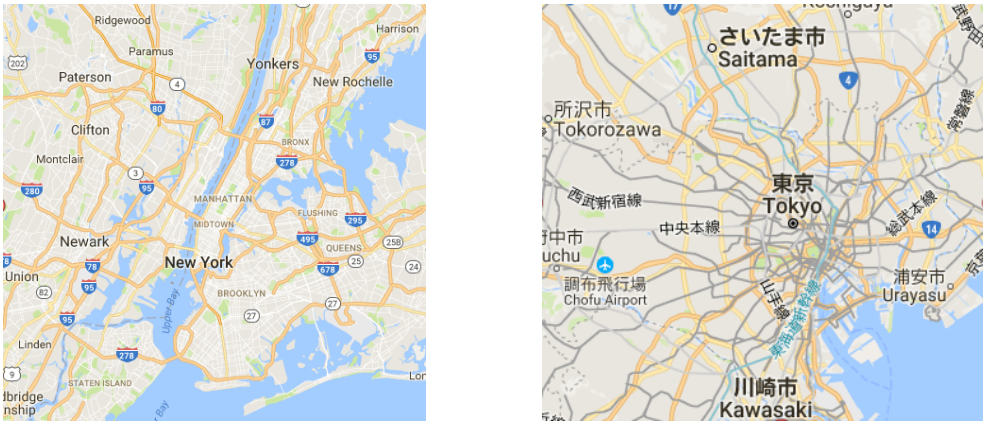


Figure 4.8: An example execution of clustering-based data generation algorithm.

Figure 4.8 illustrates the algorithm with an example. Let the parameter k be 20. The algorithm first sets K to 10 and creates 10 clusters. Since there are clusters less than 20 users, the algorithm merges R_8 and R_9 in the first iteration. After merging R_6 and R_7, all clusters have more than or equal to 20 users to be generated. Then, the algorithm continues with the Step 8 to merge the regions whose area are less than θ. Let the area of region R_5 be smaller than θ. It is then merged with R_2 which is the neighbor of R_5 with the smallest area. At the end, synthetic data generation is applied within each of the 7 regions, each with at least 20 users.

4.3 Experimental Results

In our experiments, we used datasets [51] containing 227,428 check-ins in New York City and 573,703 check-ins in Tokyo collected from Foursquare from 12 April



(a) New York City (b) Tokyo

Figure 4.9: The regions covering all user locations on map in the experiments.

2012 to 16 February 2013. Each check-in in the datasets contains time stamp, GPS coordinates, and venue information. We only used GPS coordinates and we considered each check-in as a separate user. Hence, there are 227,428 users in $\mathcal{U}_{NYC}$ and 573,703 users in $\mathcal{U}_{TKY}$. For existing facilities, we used the locations of 97 McDonald's restaurants in New York ($\mathcal{F}_{NYC}$) and 76 Yoshinoya restaurants in Tokyo ($\mathcal{F}_{TKY}$). Figure 4.9a and 4.9b show the whole regions containing user locations on map for New York City and Tokyo, respectively. We divided the whole region into a 10x10 grid for each city and selected the center of each grid as a candidate location for the new facility. We removed the candidates that are in empty regions (e.g. seas). Hence, $\mathcal{P}_{NYC}$ and $\mathcal{P}_{TKY}$ contain 69 and 72 candidate locations, respectively.

In Section 4.3.1, we present the experimental results for the proposed optimal location predictor which is explained in Section 4.1. Then, the experimental results for the server side data generation techniques are given in Section 4.3.2.

4.3.1 Synthetic Data at Client Side

We implemented our predictor to evaluate its accuracy for max-inf optimal location query and min-dist optimal location query. Initially, we executed these queries using real user locations ($\mathcal{U}_{NYC}$ and $\mathcal{U}_{TKY}$) and we ranked all candidate

locations ($\mathcal{P}_{NYC}$ and $\mathcal{P}_{TKY}$) with respect to their optimalities. We determined the best candidates for max-inf optimal location query and min-dist optimal location query. Let r_i be the ranking of the candidate location p_i when real user location data is used. To observe the accuracy of the predictor, we counted the total number of users attracted by each existing facility in $\mathcal{F}_{NYC}$ and $\mathcal{F}_{TKY}$. We provided these values ($BRNN(f)$ for each facility $f \in \mathcal{F}_{NYC}$ and $f \in \mathcal{F}_{TKY}$) to the predictor together with auxiliary information. The data generator produced synthetic user locations ($\mathcal{U}'_{NYC}$ and $\mathcal{U}'_{TKY}$) and we observed the rankings of the candidate locations when synthetic data is used in optimal location queries. Let r'_i be the ranking of the candidate location p_i returned from the predictor. We evaluate the accuracy of the predictor by measuring the standard deviation of the rankings with the following formula:

$$\sqrt{\frac{\sum_{i=1}^{|\mathcal{P}|}(r_i - r'_i)^2}{|\mathcal{P}|}}$$

where $|\mathcal{P}|$ is the number of candidates. We ran the predictor several times to show the effect of auxiliary information on the accuracy of the predictor. We also ran the predictor with different ω values to observe the effect of ω on accuracy. We present the evaluation results for max-inf optimal location query and min-dist optimal location query in Section 4.3.1.1 and 4.3.1.2, respectively. For each query type, we firstly present the results for $\omega = 0.5$ and then show the effect of ω on the accuracy.

We also illustrate the ranking of the candidates with figures. In these figures, the red plus signs represent the existing facilities ($\mathcal{F}_{NYC}$ and $\mathcal{F}_{TKY}$), the gray circles represent the users ($\mathcal{U}_{NYC}$ and $\mathcal{U}_{TKY}$), the blue diamonds represent candidate locations ($\mathcal{P}_{NYC}$ and $\mathcal{P}_{TKY}$), and the blue lines show the boundaries of the Voronoi regions of existing facilities. We marked the best candidates with circles and second best and third best candidates with rectangles. In addition, we show the ranking of a candidate as p_i:j, in which p_i refers to a candidate location and j refers to its ranking.

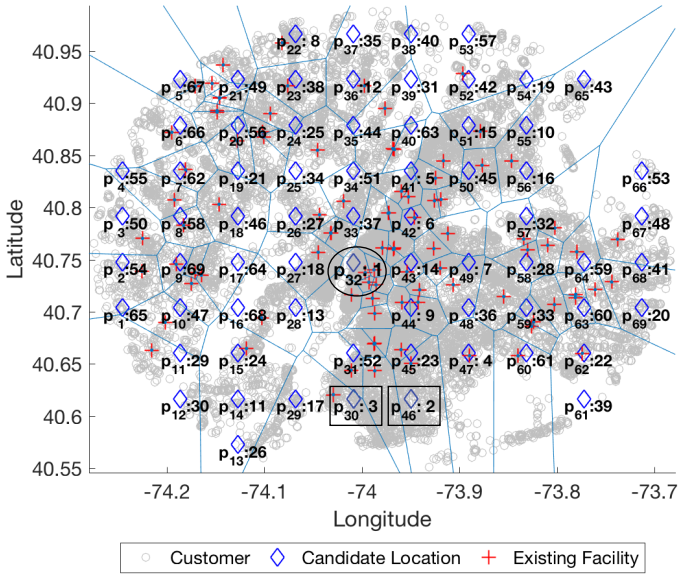


Figure 4.10: Ranking of candidate locations in NYC when real data is used in max-inf optimal location query.

4.3.1.1 Max-Inf Optimal Location Query

Max-inf optimal location query returns a candidate location p_i which maximizes the total number of users attracted by the new facility if it is built at the location p_i. Figure 4.10 and 4.11 show the rankings of the candidate locations in $\mathcal{P}_{NYC}$ and $\mathcal{P}_{TKY}$ when real user locations ($\mathcal{U}_{NYC}$ and $\mathcal{U}_{TKY}$) are used.

In New York City, the best candidate for maximizing the total number of users attracted by the new facility is p_{32} as shown in Figure 4.10. It attracts 5,341 users. The other candidates in top five are p_{46}, p_{30}, p_{47}, and p_{41}, and the total number of users attracted by these candidates are 4,599, 3,551, 3,321, and 3,025, respectively.

In Tokyo, the best candidate returned from max-inf optimal location query is p_{53} and the total number of users attracted by the new facility is 42,411 if it is built at the location p_{53}. The other candidates in top five are p_{32}, p_9, p_{46}, and p_{55}, and the total number of users attracted by these candidates are 13,528, 13,384, 13,338, and 10,458, respectively.

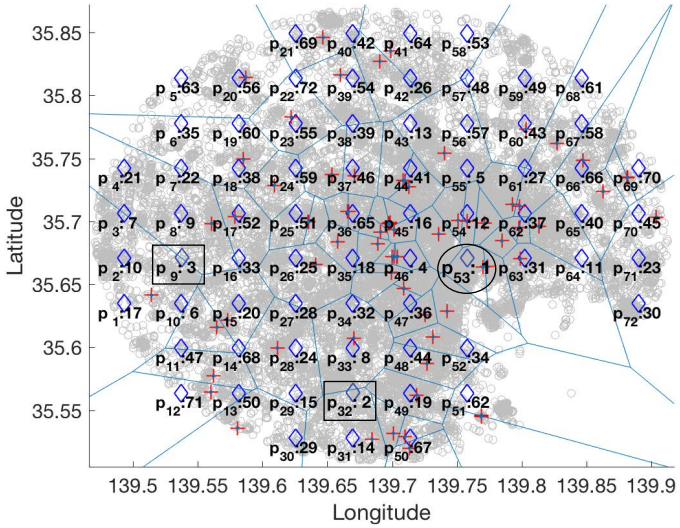


Figure 4.11: Ranking of candidate locations in Tokyo when real data is used in max-inf optimal location query.

In the evaluation of the predictor, we provided $BRNN(f)$ for each facility $f \in \mathcal{F}_{NYC}$ and $f \in \mathcal{F}_{TKY}$ to the predictor. Figure 4.12 and 4.13 show the rankings when minimum and maximum coordinates (i.e. AI 1) are also provided to the predictor. For both cities, the predictor returns the same best candidate with the knowledge of AI 1. The predictor estimates the total number of users attracted by p_{32} as 11,357 in New York City and the total number of users attracted by p_{53} as 33,989 in Tokyo. In New York City, the predictor also finds the same second best candidate correctly. The standard deviations in the rankings for New York City and Tokyo are 14.4272 and 12.9271, respectively.

When we also provide AI 2 and AI 3 to the predictor, it still returns the same best candidates as shown in Figure 4.14 and 4.15. Moreover, using AI 2 and AI 3 decreases the standard deviation of the rankings. The standard deviation decreases from 14.4272 to 12.2451 in New York and decreases from 12.9271 to 11.6583 in Tokyo. This result indicates that providing more information to the predictor improves the accuracy in the rankings, as expected.

To experiment with the case where the locations of some users are known (i.e. AI 4), we provided varying number of user locations to the predictor. As evident in Figure 4.16a, increasing the ratio of known users decreases the standard

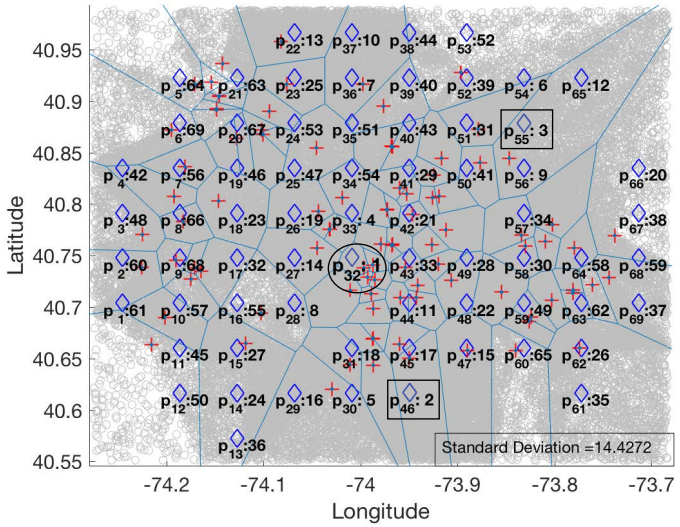


Figure 4.12: Ranking of candidate locations in NYC when the optimal location predictor uses AI 1 in max-inf optimal location query.

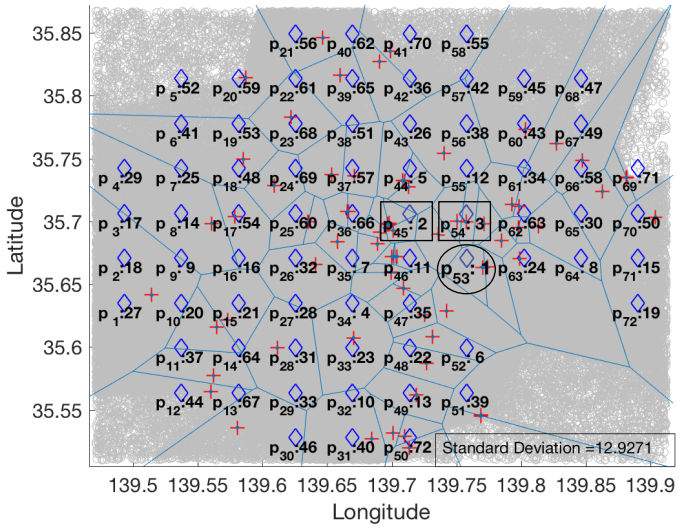


Figure 4.13: Ranking of candidate locations in Tokyo when the optimal location predictor uses AI 1 in max-inf optimal location query.

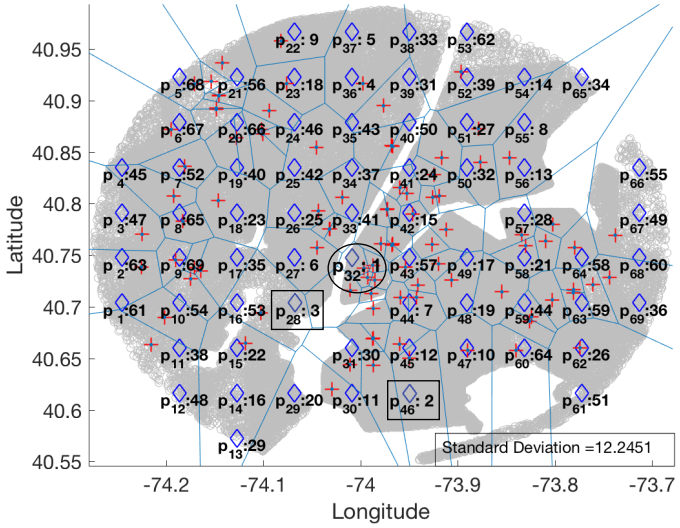


Figure 4.14: Ranking of candidate locations in NYC when the optimal location predictor uses AI 2 and AI 3 in max-inf optimal location query.

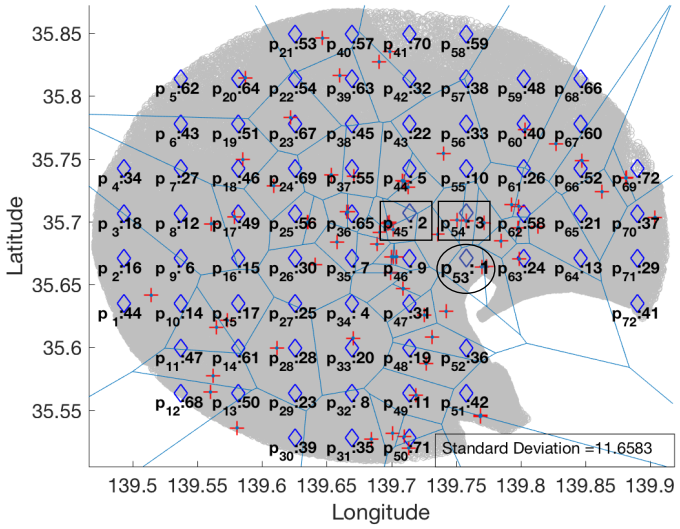
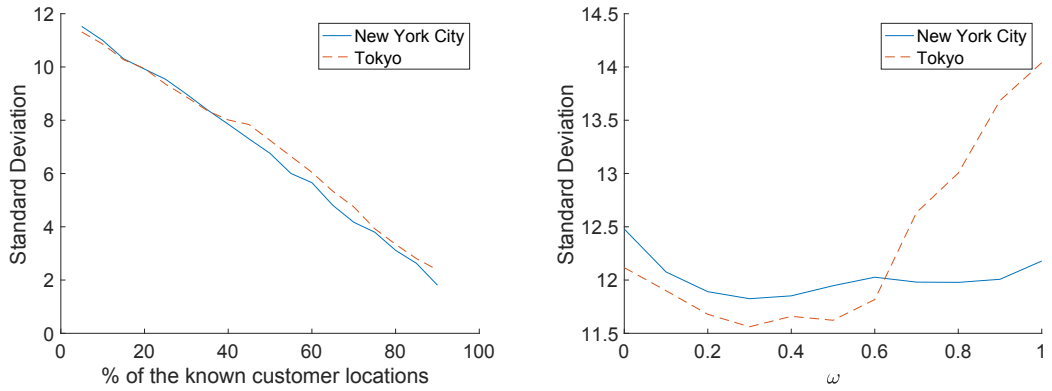


Figure 4.15: Ranking of candidate locations in Tokyo when the optimal location predictor uses AI 2 and AI 3 in max-inf optimal location query.



(a) Impact of AI 4 (b) Impact of ω

Figure 4.16: Impact of AI 4 and ω on the standard deviation of rankings in max-inf optimal location query.

deviation of the rankings. For instance, when 50% of the user locations are known, the standard deviation decreases to nearly 7 in both cities.

We also conducted experiments to observe the impact of ω on the standard deviation. As it is mentioned in Section 4.1.2, when ω is equal to 0 the distribution is only based on the areas of the triangles. Hence, we use $\omega = 0$ as the baseline which provides a distribution in Voronoi region that is similar to uniform distribution. Figure 4.16b shows the standard deviation for different values of ω between 0 and 1 when AI 2 and AI 3 are provided to the predictor. For both cities, minimum standard deviation is obtained when ω is selected as 0.3. The standard deviation is 11.8248 in New York City and 11.5614 in Tokyo when ω is equal to 0.3. We also analyzed the rankings and we observed that the predictor's top five candidates are same for $\omega = 0.3$ and $\omega = 0.5$. As evident in Figure 4.16b, best accuracy is achieved when ω value is in the range of $[0.2, 0.5]$. The standard deviation is lower than the baseline ($\omega = 0$) when ω is selected in this range.

To evaluate the accuracy of the predictor when no AI is known, we provided larger regions than the minimum bounding rectangle (i.e. AI 1). We expanded the height and width of the minimum bounding rectangle iteratively and Figure 4.17 shows the standard deviation for different expansion percentages. For instance, when we provided a rectangular region whose height and width are 20% greater than the minimum bounding rectangle, the standard deviation increases to nearly

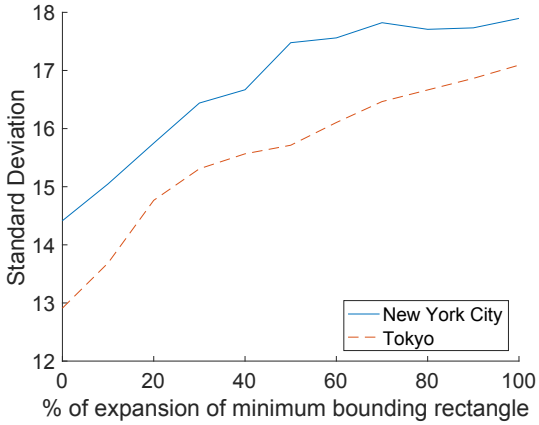


Figure 4.17: Standard deviation of rankings when no AI is provided to the optimal location predictor in max-inf optimal location query.

15 in Tokyo and nearly 16 in New York City. As expected, providing larger regions increases the standard deviation. However, the predictor returns the same best candidates for both cities without using auxiliary information, because max-inf optimal location query returns the candidate which attracts maximum amount of users without considering the distances from users to their nearest facilities. Therefore, the effect of generating users outside the minimum bounding rectangle on the best candidate is low in max-inf optimal location query.

4.3.1.2 Min-Dist Optimal Location Query

Min-dist optimal location query returns a candidate location p_i which minimizes the average distance between each user and her nearest facility if the new facility is built at the location p_i. We conducted the same set of experiments for this query as well. Figure 4.18 shows the ranking of candidates ($\mathcal{P}_{NYC}$) in NYC when the real user locations ($\mathcal{U}_{NYC}$) are used in min-dist optimal location query. In New York City, the average distance of users to their nearest facilities are minimized if the new facility is built at p_{46}. The average distance becomes 1.4433 km if p_{46} is selected as the location of the new facility. The other candidates in top five are p_{65}, p_{14}, p_{30}, and p_{54}, and building a new facility at these locations decreases the average distances to 1.4567 km, 1.4587 km, 1.4588 km, and 1.4611 km, respectively.

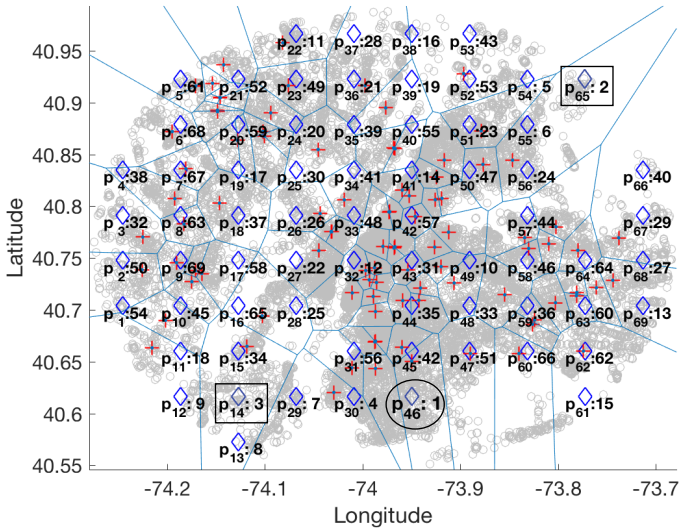


Figure 4.18: Ranking of candidate locations in NYC when real data is used in min-dist optimal location query.

The ranking of candidates ($\mathcal{P}_{TKY}$) in Tokyo is given in Figure 4.19 when the real user locations ($\mathcal{U}_{TKY}$) are used. In Tokyo, the best candidate for min-dist optimal location query is p_3. The average distance becomes 1.3346 km, if the new facility is built at p_3. The other candidates in top five are p_2, p_{32}, p_{72}, and p_{53}, and building a new facility at these locations decreases the average distances to 1.3431 km, 1.3436 km, 1.346 km, and 1.3475 km, respectively.

Figure 4.20 and 4.21 show the rankings for both cities according to the predictor with only AI 1. In New York City, the predictor returns p_{65} as the best candidate, which is actually the second best candidate as shown in Figure 4.18. The real best candidate (p_{46}) is ranked third by the predictor. In Tokyo, the predictor returns p_{72} as the best candidate; however, its actual rank is 5. The real best candidate (p_3) is ranked second by the predictor. The standard deviation is 12.7632 in New York City and 12.5266 in Tokyo, when only AI 1 is provided to the predictor.

For min-dist optimal location query, only AI 1 is not sufficient for the predictor to return the same best candidate. Since the predictor only uses AI 1, it generates users in empty areas such as seas. Therefore, distance from a user to her nearest facility is usually higher than the real one, which affects the accuracy considerably.

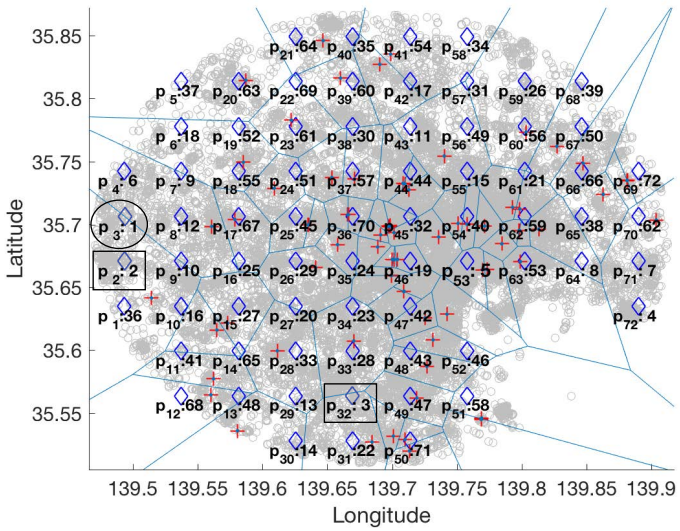


Figure 4.19: Ranking of candidate locations in Tokyo when real data is used in min-dist optimal location query.

AI 2 and AI 3 should be provided to the predictor to achieve a better accuracy.

Figure 4.22 and 4.23 show the rankings according to the predictor, when we provided AI 2 and AI 3 to the predictor. It found the same best candidates for both New York City and Tokyo. The average distance values are closer to the real values, when the predictor uses AI 2 and AI 3. The standard deviation also decreases from 12.7632 to 11.0362 in New York and decreases from 12.5266 to 10.1009 in Tokyo.

Similar to max-inf optimal location query, the standard deviation of the rankings is inversely proportional to the ratio of known user locations (i.e. AI 4). Standard deviation for different values of percentage of known user locations is given in Figure 4.24a. As evident in Figure 4.24a, the accuracy of the predictor increases when the locations of more users are provided to the predictor.

Figure 4.24b shows the standard deviation for different values of ω between 0 and 1 when AI 2 and AI 3 are provided to the predictor in min-dist optimal location query. In New York City, minimum standard deviation (10.3881) is obtained when $\omega = 0.4$. In Tokyo, standard deviation is minimum (9.9163) when $\omega = 0.2$. In both cities, the best accuracy is achieved when ω value varies

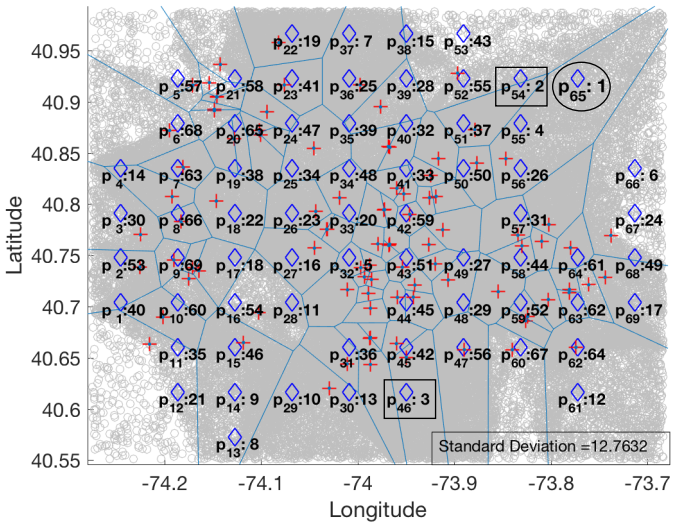


Figure 4.20: Ranking of candidate locations in NYC when the optimal location predictor uses AI 1 in min-dist optimal location query.

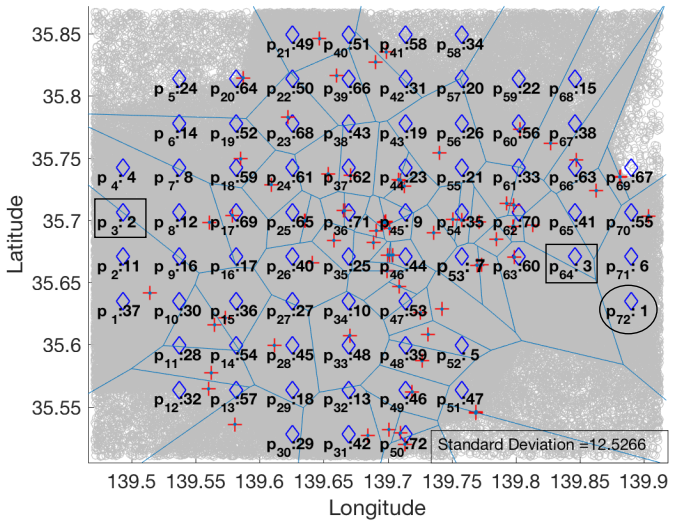


Figure 4.21: Ranking of candidate locations in Tokyo when the optimal location predictor uses AI 1 in min-dist optimal location query.

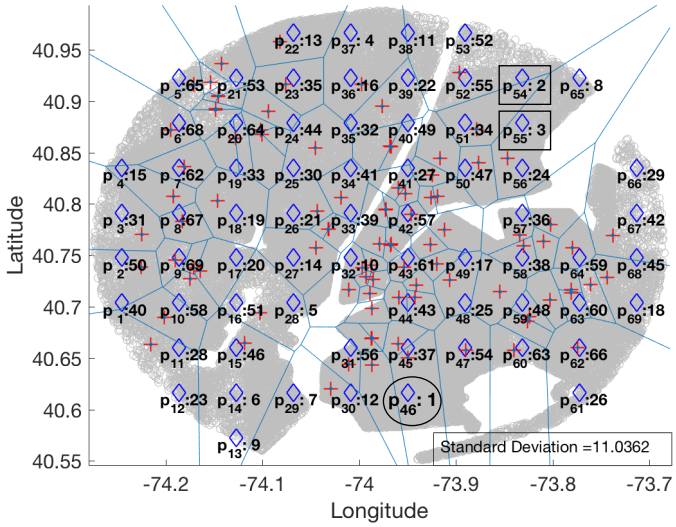


Figure 4.22: Ranking of candidate locations in NYC when the optimal location predictor uses AI 2 and AI 3 in min-dist optimal location query.

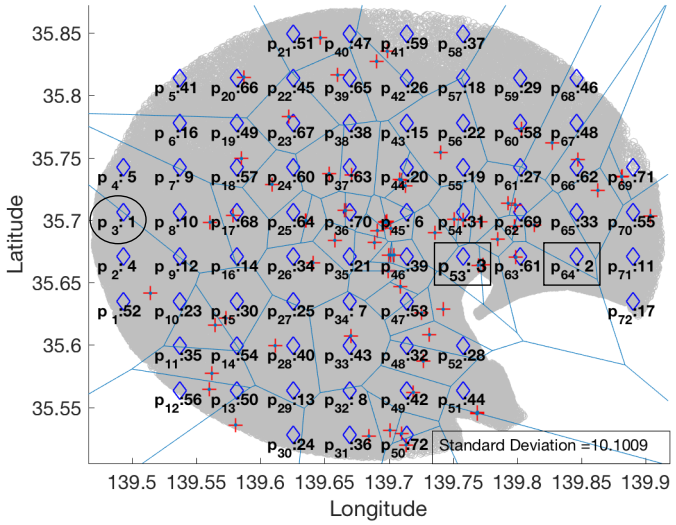
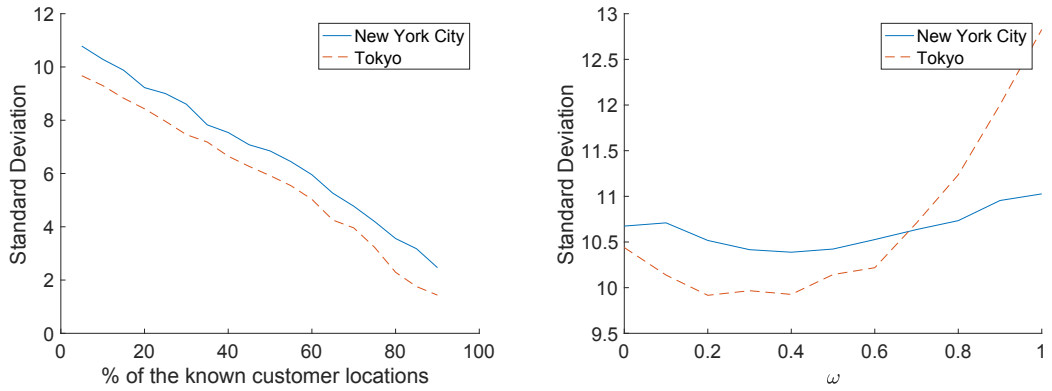


Figure 4.23: Ranking of candidate locations in Tokyo when the optimal location predictor uses AI 2 and AI 3 in min-dist optimal location query.



(a) Impact of AI 4 (b) Impact of ω

Figure 4.24: Impact of AI 4 and ω on the standard deviation of rankings in min-dist optimal location query.

between 0.2 and 0.5. Similar to max-inf optimal location query, selecting ω value in the range of $[0.2, 0.5]$ provides better accuracy than the baseline ($\omega = 0$). Moreover, when we analyze the rankings of candidate locations, the predictor's top five candidates are same for all ω values in this range. Therefore, ω should be selected between 0.2 and 0.5 to improve accuracy.

Figure 4.25 depicts the standard deviation of the rankings when the given region to the predictor is larger than the minimum bounding rectangle. As in max-inf optimal location query, the standard deviation increases when the size of the region increases. Unlike max-inf optimal location query, the predictor does not return the same best candidates when no auxiliary information is provided. Therefore, providing auxiliary information in min-dist optimal location query is more important than max-inf optimal location query to find the same best candidate.

4.3.2 Synthetic Data at Server Side

To evaluate the quality of differentially private synthetic data, we conducted experiments with the same setting given in Section 4.3. Particularly, we compare the quality of data for the grid-based approach and the clustering-based approach in New York City. We use two different values of k as 100 and 1000 in order to

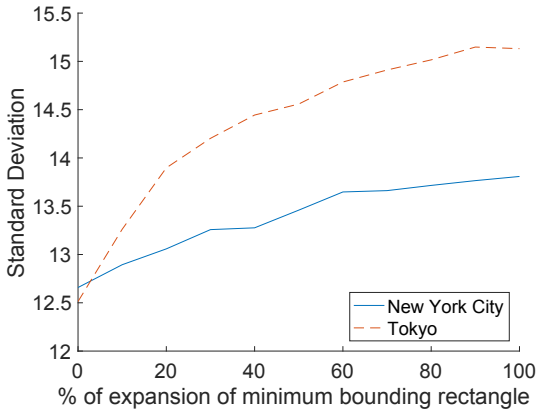


Figure 4.25: Standard deviation of rankings when no AI is provided to the optimal location predictor in min-dist optimal location query.

observe the effect of k on quality. We first compare the similarity of original data with synthetic data produced by grid-based and clustering-based algorithms using Dice and Jaccard coefficients. For two databases D and D', the Dice coefficient is measured as

$$2 \cdot \frac{|D \bigcap D'|}{|D| + |D'|},$$

and the Jaccard coefficient is measured as

$$\frac{|D \bigcap D'|}{|D| + |D'| - |D \bigcap D'|}.$$

To measure similarity for location data, we divided the whole region covering all users into 1000×1000 grid and counted the number of users in each grid cell. Then, we measured Dice and Jaccard coefficients using the number of users in each cell. Table 4.2 shows the similarities for grid-based approach and clustering-based approach. As evident in Table 4.2, clustering-based approach produces more similar synthetic data than grid-based approach. The similarity values for clustering-based approach is two or three times greater than grid-based approach for different k values and similarity coefficients.

Similar to Section 4.3, we also compare the rankings of the candidates for synthetic and real data. We first execute optimal location queries with real data and then compare the results when synthetic data is used. We present the results for max-inf optimal location query and min-dist optimal location query. Along

Table 4.2: Similarity of real data and synthetic data for different k values and similarity coefficients.

	Grid-based	Clustering-based
$k = 100$, Dice	0.082246	0.289867
$k = 100$, Jaccard	0.042886	0.169500
$k = 1000$, Dice	0.082480	0.238578
$k = 1000$, Jaccard	0.043014	0.135446

with the standard deviation, we measure the Normalized Discounted Cumulative Gain (NDCG@10) and the Precision@10 for both approaches to evaluate rankings. NDCG is used for measuring ranking quality. Discounted cumulative gain at rank 10 is calculated as

$$rel_1 + \frac{rel_2}{log_2 2} + \frac{rel_3}{log_2 3} + ... + \frac{rel_{10}}{log_2 10}$$

where rel_i is the relevance associated with the result at rank i. We use the number of users attracted by the new facility as rel_i in max-inf query and the average distance from each user to her nearest facility as rel_i in min-dist query. NDCG@10 is the ratio of DCG@10 to best possible DCG@10. NDGC produces values on the interval 0 to 1 to make comparisons easier. Precision@k is another useful metric to compute the ratio of relevant results in top k. We measure Precision@10 by dividing the number of relevant results in top 10 by 10. A result is relevant if it is in top 10 in the rankings when real data is used.

For the grid-based approach, we divided the city into a 25*25 grid. Hence, the number of grid cells m is equal to 625 in our experiments. For clustering-based approach, we selected the number of clusters K as 2000 at the beginning of the algorithm. Moreover, we set the the threshold θ for minimum subregion size to 1 km^2. Therefore, the area of each region is greater than 1 km^2 at the end of the algorithm. We present the results for two different k values such as 100 and 1000. In Section 4.3.2.1 and 4.3.2.2, experiment results are given for max-inf and min-dist queries, respectively.

4.3.2.1 Max-Inf Optimal Location Query

Since we use the same setting with the experiments in Section 4.3.1, the rankings of the candidate locations when real user locations are used are same with the rankings shown in Figure 4.10. Figure 4.26 shows the rankings in New York City when k is 100 and the grid-based algorithm is used in data generation. The top three candidates are same with the real top three candidates. The standard deviation is 5.811, NDCG@10 is 0.950235, and Precision@10 is 0.7. When clustering-based algorithm is used the standard deviation is 4.6188, NDCG is 0.990718, and Precision@10 is 0.9. In addition, as evident in Figure 4.27, the top four candidates are same with the real top four candidates in clustering-based approach.

Clustering-based approach also outperforms grid-based approach in terms of ranking quality when k is selected as 1000. The results for $k = 1000$ are presented in Figure 4.28 and 4.29. In grid-based data generation, the standard deviation is 9.8936, NDCG@10 is 0.893733, and Precision@10 is 0.6. In clustering-based data generation, the standard deviation is 13.0228, NDCG@10 is 0.931646, and Precision@10 is 0.7. Although standard deviation is better in grid-based approach, NDCG@10 and Precision@10 are better in clustering-based approach. Moreover, the ranking quality decreases when k increases, as expected.

4.3.2.2 Min-Dist Optimal Location Query

In min-dist optimal location query, the performance of the clustering-based approach and grid-based approach are similar in terms of ranking quality metrics. Figure 4.18 shows the ranking of candidates in NYC when the real customer locations are used in min-dist optimal location query. The results are presented in Figure 4.30 and 4.31, when k is selected as 100. In grid-based approach, the standard deviation is 5.116, NDCG@10 is 0.999018, and Precision@10 is 0.9. When clustering-based algorithm is used the standard deviation is 4.2255, NDCG is 0.998989, and Precision@10 is 0.9. In addition, Figure 4.32 and 4.33 also show

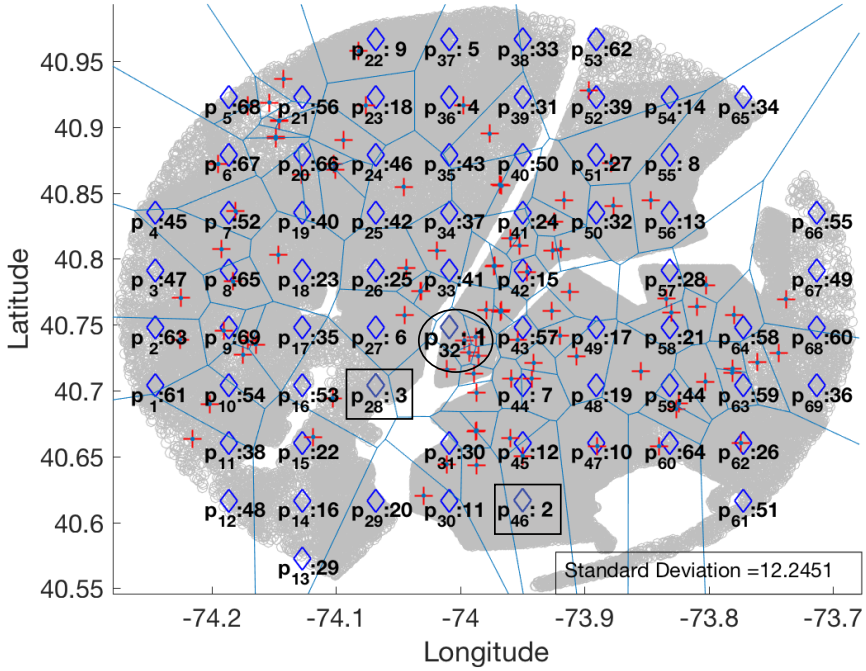


Figure 4.26: Ranking of candidate locations in NYC when $k = 100$ and synthetic data produced by the grid-based algorithm is used in max-inf optimal location query.

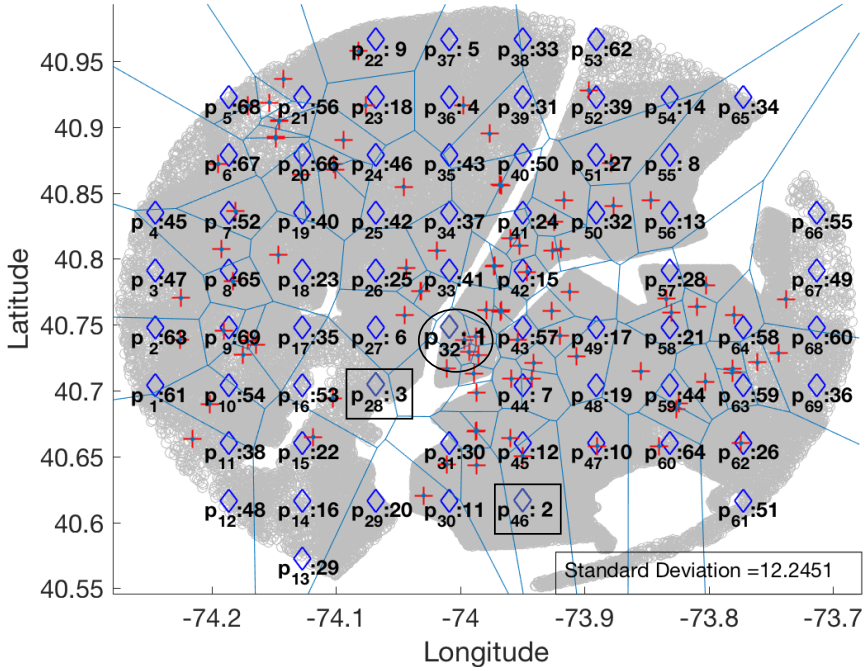


Figure 4.27: Ranking of candidate locations in NYC when $k = 100$ and synthetic data produced by the clustering-based algorithm is used in max-inf optimal location query.

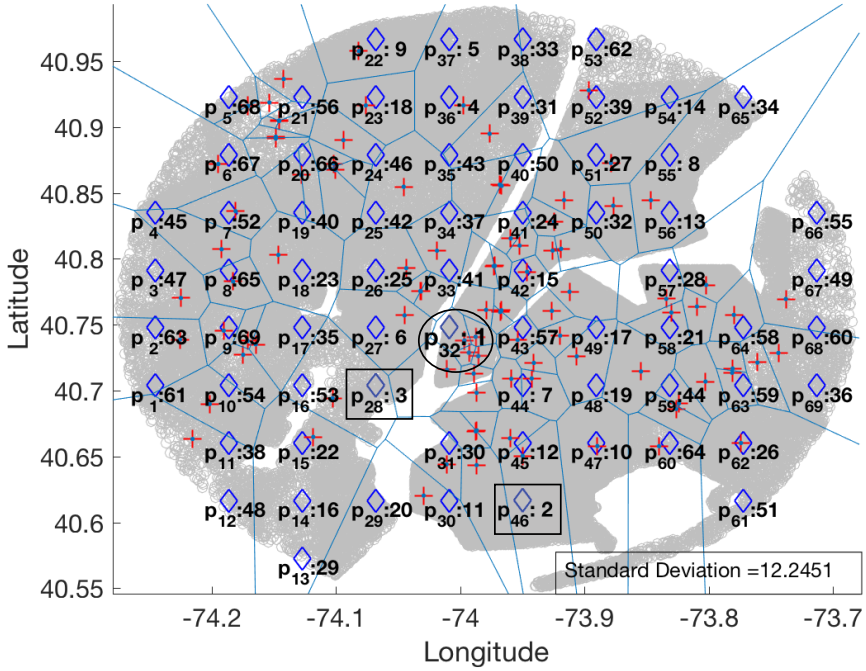


Figure 4.28: Ranking of candidate locations in NYC when $k = 1000$ and synthetic data produced by the grid-based algorithm is used in max-inf optimal location query.

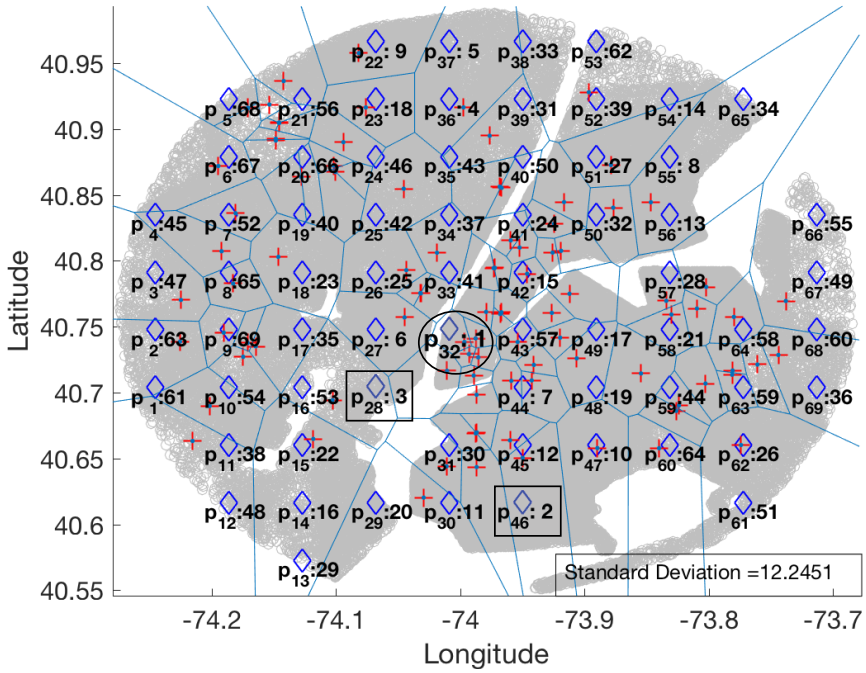


Figure 4.29: Ranking of candidate locations in NYC when $k = 1000$ and synthetic data produced by the clustering-based algorithm is used in max-inf optimal location query.

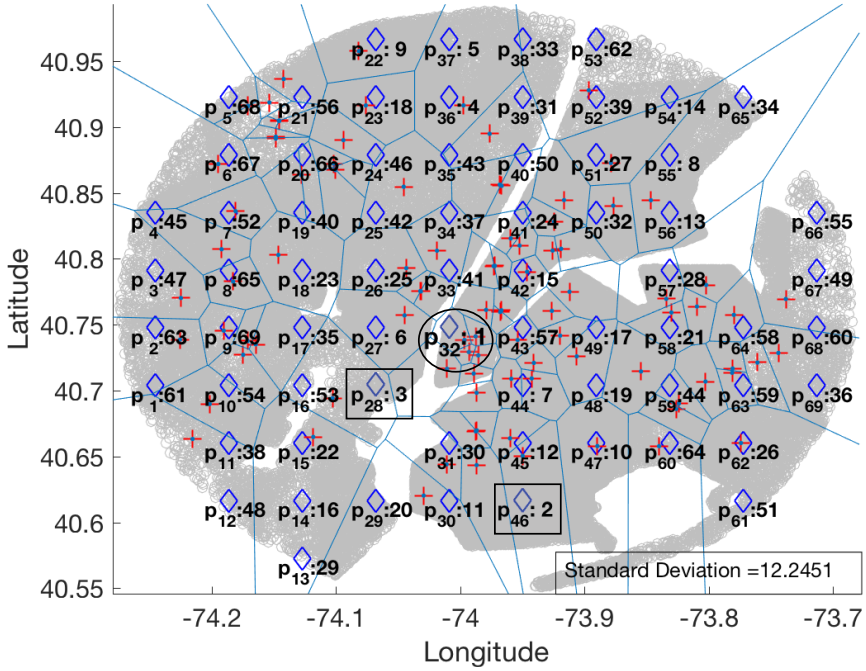


Figure 4.30: Ranking of candidate locations in NYC when $k = 100$ and synthetic data produced by the grid-based algorithm is used in min-dist optimal location query.

the rankings for $k = 1000$. When k is 1000 and grid-based approach is used the real best candidate is not found. The standard deviation is 10.1952, NDCG@10 is 0.999018, and Precision@10 is 0.8. However, the real best candidate is found in clustering-based approach. The standard deviation is 10.8921, NDCG@10 is 0.998715, and Precision@10 is 0.8 in clustering-based approach.

4.4 Conclusion

In this chapter, we first proposed an optimal location predictor which does not require the user locations. By using the density of the users in each existing facility and the given auxiliary information, it returns a candidate location from a set of candidates. After generating user locations based on given information, the predictor runs a query for finding the best location. During data generation, the predictor does not simply distribute users uniformly, it considers the density

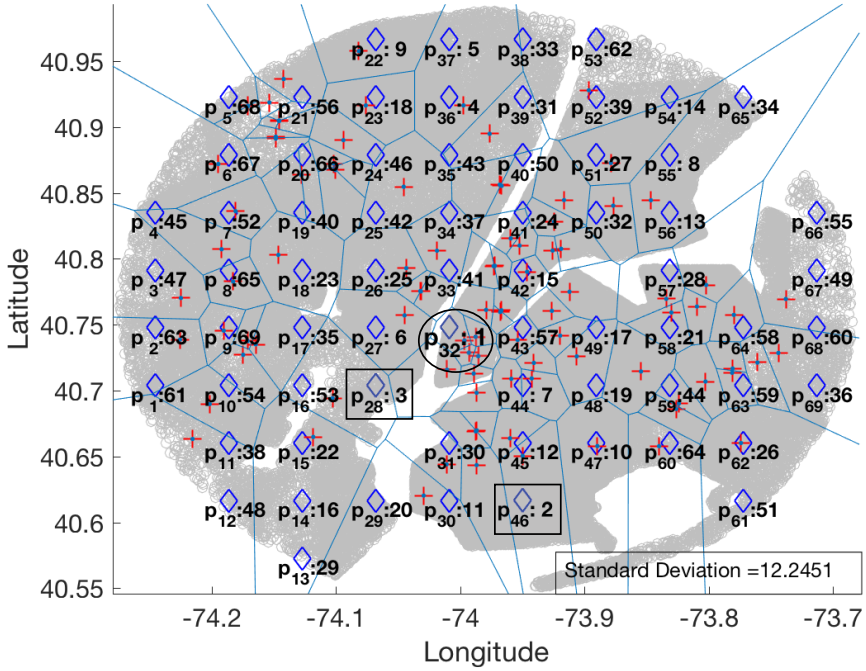


Figure 4.31: Ranking of candidate locations in NYC when $k = 100$ and synthetic data produced by the clustering-based algorithm is used in min-dist optimal location query.

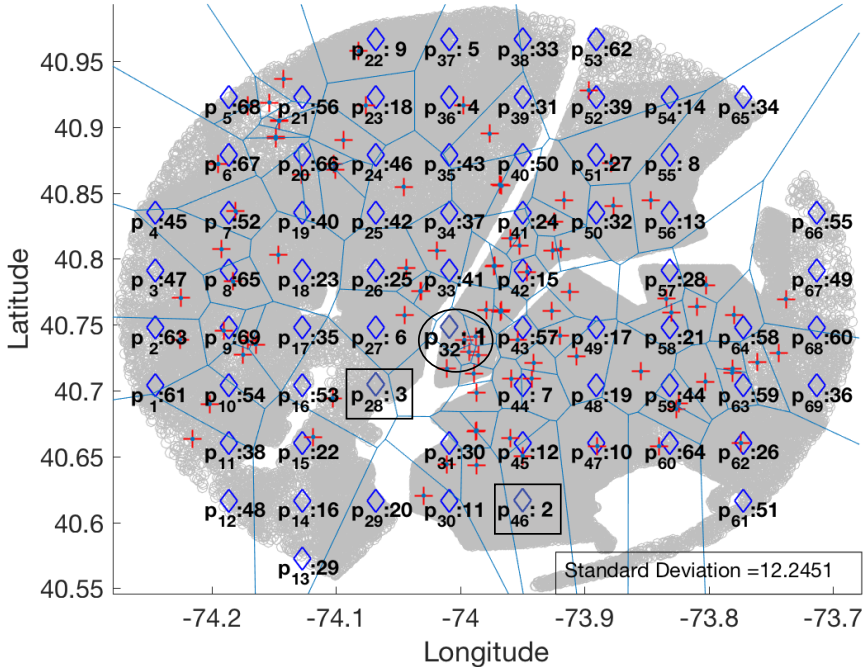


Figure 4.32: Ranking of candidate locations in NYC when $k = 1000$ and synthetic data produced by the grid-based algorithm is used in min-dist optimal location query.

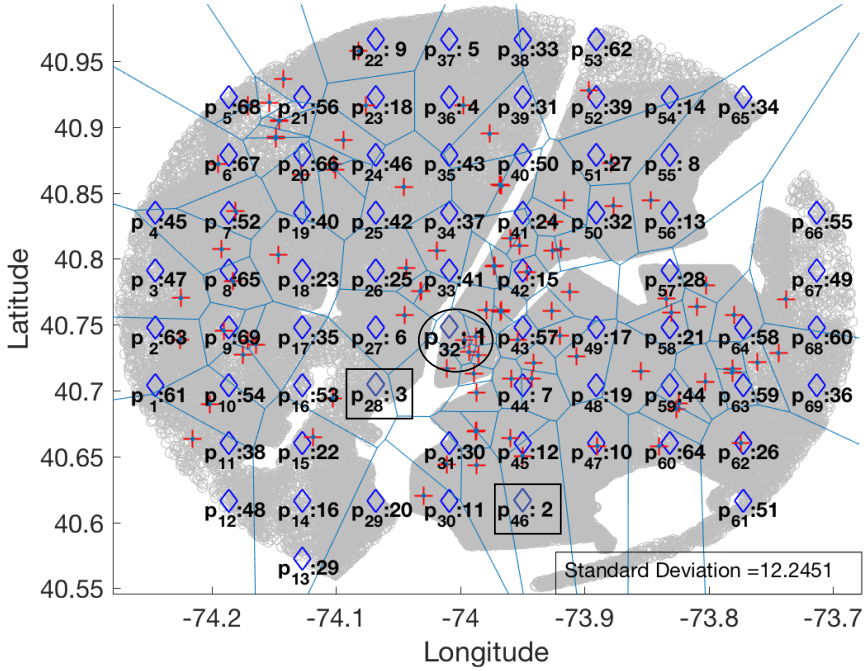


Figure 4.33: Ranking of candidate locations in NYC when $k = 1000$ and synthetic data produced by the clustering-based algorithm is used in min-dist optimal location query.

of users in neighbor facilities that are selected from the Voronoi diagram of the facilities. Two facilities are neighbors of each other, if their Voronoi regions share a common edge. Hence, the data generator divides the Voronoi region of each existing facility into smaller triangular regions and generates user locations in each smaller region. We performed experiments with real datasets to evaluate the accuracy of the optimal location predictor. The predictor found the real best candidate in both max-inf and min-dist optimal location queries when the convex hull of user locations (i.e. minimum bounding polygon) and the empty regions in the cities are given. Hence, it is useful to know the boundaries of the region containing user locations to obtain accurate results. In addition, our experiment results indicate that ω value used in data generation should be selected between 0.2 and 0.5 to achieve high accuracy.

Since the predictor generates location data randomly, it may not return the best candidate in the following cases:

- if the difference of optimality scores of top two candidates is low. The optimality score of a candidate p_i is calculated as $|\mathcal{I}(p_i)|$ in max-inf optimal location query, and $\mathcal{A}(p_i)$ in min-dist optimal location query. For instance, in max-inf optimal location query, if the best candidate attracts 350 users and the second best candidate attracts 348 users, the predictor may not return the real best candidate.
- if the total number of existing facilities (i.e. $|\mathcal{F}|$) is low.
- if the existing facilities have a highly skewed distribution.

In such cases, knowing the locations of some users by businesses increases the chance of returning the best one. The experiment results show that providing more information improves the accuracy of the predictor. The proposed predictor facilitates running optimal location queries by businesses without knowing their users' locations.

We also developed methods to generate privacy-preserving synthetic location data by the server. We first presented a grid-based method as baseline algorithm. Then, we presented our clustering-based method for synthetic data generation. Both of the approaches satisfy differential privacy. Our experimental results show that the similarity of synthetic data and real data are higher in clustering-based method. Moreover, the accuracy of analytics results are better in clustering-based method against grid-based method. Location data owners can use the proposed data generation methods for publishing a private and synthetic copy of their databases.

Chapter 5

Conclusion and Future Work

Businesses use data analytics tools for a wide range of business decisions ranging from operational decisions to strategic decisions. One critical decision made by a business is choosing a location for its new facility by analyzing the locations of its customers. In the literature, there are several solutions to find the best location for the new facility considering different objective functions. These solutions work when the locations of customers are given. However, businesses do not know the locations of their customers most of the time. This valuable data is typically stored by the mobile operators and location-enhanced service providers. Since location data contains sensitive information about individuals, it is not possible to share it with businesses. Hence, privacy-preserving solutions are needed by businesses when they want to do location analytics in the absence of customer locations.

In this dissertation, we proposed two complementary approaches for the problem and illustrated these approaches in the context of optimal location selection problem. First, we proposed privacy-preserving query processing protocols that enable collaborative analytics. With the proposed protocols, location-based queries can be answered by data owners without sharing their data with other businesses and without accessing sensitive information such as the customer list

of the businesses that send the query. In the proposed query processing protocols, homomorphic encryption is used for keeping sensitive data hidden from unauthorized parties. Since the defined location-based queries return aggregate results, differential privacy is also satisfied to prevent information leak about any individual in the database. The performance evaluations show that the proposed protocols are practical, efficient, and scalable.

The second approach to do analytics without location data is generating synthetic location data and using it in location analytics. Although many businesses do not know exact locations of their customers, they know partial information about customer locations such as the density of customers in each existing facility. The partial information may also be retrieved as a result of proposed query processing protocols. We proposed an optimal location predictor that first produces synthetic location data based on partial information known by businesses and then executes optimal location queries over generated location data. The proposed data generator partitions the region as a Voronoi diagram and determines the number of customers in each subregion using the density of the customers in the neighbor subregions. The experiment results show that optimal location for a new facility can be found among several candidates when optimal location queries are executed on generated location data based on the partial information.

Location data owners can also generate synthetic location data as an alternative to data anonymization and perturbation. Generated data can be shared with businesses and it can also be used in location analytics. To protect the privacy of individuals, synthetic data should satisfy differential privacy. We proposed two types of algorithms based on grid-based and clustering-based partitioning. While both methods produce privacy-preserving and useful synthetic data, our experiments show that the clustering-based method outperforms the grid-based approach in terms of representation accuracy. Our experimental results also show that the accuracy of the analytics is better for synthetic data generated at the server side against synthetic data generated at the client side.

The proposed solutions can be used when businesses do not know the customer locations. In the future, our work can be extended in the following directions:

- A follow up work is to support competitive facility location problem. In our work, we considered all existing facilities as the facility of a business. In practice, there are also facilities of the competitors and these facilities affect the selection of the optimal location. Businesses pay attention to attract people from competitors instead of attracting customers from their own facilities. Therefore, more than one categories of existing facilities can be considered as a future work.

- Another follow up work is to develop solutions for privacy-preserving query processing to analyze location data in multiple data owners. In our current solution, there is a single server $\mathcal{S}$ as the data owner. However, location data may be owned by several data owners. When these data owners have common users, the problem of collaborative analytics becomes interesting and challenging. Hence, our secure two-party solutions can be extended to secure multi-party solutions.

- In our work, each customer has a unique location information. However, location data owners collect user locations at different time periods. These spatio-temporal records are frequently used in business intelligence. Combining space and time dimensions provide more opportunities to businesses for deriving valuable information. Developing privacy-preserving solutions to analyze spatio-temporal data in location data owners is a possible research direction.

- In Chapter 4, we presented a data generator which uses partial information known by a business. The proposed approach can be applied to different optimization problems when data is not available. If there is partial information about data such as the number of items in different clusters, synthetic data can be generated similarly and it can be used in optimization. Hence, generating synthetic data for different optimization problems and evaluating their optimization performance is a potential follow up of this work. Another follow up work is to apply bootstrap methods for data generation and evaluating their accuracy for the case where the locations of some customers are known. These methods allow increasing the data size by generating new samples based on the original samples. Therefore,

bootstrap methods for spatial data [52, 53] can also be potentially used for data generation if a subset of customer locations is known.

Bibliography

[1] L. Garber, "Analytics goes on location with new approaches," *Computer*, no. 4, pp. 14–17, 2013.

[2] Y. Du, D. Zhang, and T. Xia, "The optimal-location query," in *Advances in Spatial and Temporal Databases*, pp. 163–180, Springer, 2005.

[3] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive computation of the min-dist optimal-location query," in *Proceedings of the 32nd international conference on Very large data bases*, pp. 643–654, VLDB Endowment, 2006.

[4] J. Qi, R. Zhang, Y. Wang, A. Y. Xue, G. Yu, and L. Kulik, "The min-dist location selection and facility replacement queries," *World Wide Web*, vol. 17, no. 6, pp. 1261–1293, 2014.

[5] Y.-A. De Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific reports*, vol. 3, p. 1376, 2013.

[6] E. Yilmaz, H. Ferhatosmanoglu, E. Ayday, and R. C. Aksoy, "Privacy-preserving aggregate queries for optimal location selection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[7] E. Yilmaz, S. Elbasi, and H. Ferhatosmanoglu, "Predicting optimal facility location without customer locations," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2121–2130, ACM, 2017.

[8] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 301–312, IEEE, 2004.

[9] C. Böhm and F. Krebs, "The k-nearest neighbour join: Turbo charging the kdd process," *Knowledge and Information Systems*, vol. 6, no. 6, pp. 728–749, 2004.

[10] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi, "Constrained nearest neighbor queries," in *International Symposium on Spatial and Temporal Databases*, pp. 257–276, Springer, 2001.

[11] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *ACM SIGMOD Record*, vol. 29, pp. 201–212, ACM, 2000.

[12] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu, "Efficient method for maximizing bichromatic reverse nearest neighbor," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1126–1137, 2009.

[13] F. Chen, H. Lin, Y. Gao, and D. Lu, "Capacity constrained maximizing bichromatic reverse nearest neighbor search," *Expert Systems with Applications*, vol. 43, pp. 93–108, 2016.

[14] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He, "Top-k most influential locations selection," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 2377–2380, ACM, 2011.

[15] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu, "Maxfirst for maxbrknn," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 828–839, IEEE, 2011.

[16] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long, "Efficient algorithms for optimal location queries in road networks," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 123–134, ACM, 2014.

[17] J. Cardinal and S. Langerman, "Min-max-min geometric facility location problems," in *Proc. European Workshop on Computational Geometry (EWCG'06)*, pp. 149–152, 2006.

[18] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: a review and open problems," in *Proceedings of the 2001 workshop on New security paradigms*, pp. 13–22, ACM, 2001.

[19] W. Du and M. J. Atallah, "Protocols for secure remote database access with approximate matching," in *E-Commerce Security and Privacy*, pp. 87–111, Springer, 2001.

[20] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar, "Preserving user location privacy in mobile data management infrastructures," in *Privacy Enhancing Technologies*, pp. 393–412, Springer, 2006.

[21] Z. Wu, L. Yu, J. Zhu, H. Sun, Z. Guan, and Z. Chen, "A hybrid approach for privacy preservation in location based queries," in *Web-Age Information Management*, pp. 315–326, Springer, 2013.

[22] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *Proceedings of the 32nd international conference on Very large data bases*, pp. 763–774, VLDB Endowment, 2006.

[23] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 121–132, ACM, 2008.

[24] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pp. 311–319, IEEE, 2008.

[25] X. Yi, R. Paulet, E. Bertino, and V. Varadharajan, "Practical k nearest neighbor queries with location privacy," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 640–651, IEEE, 2014.

[26] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, vol. 3876, pp. 265–284, Springer, 2006.

[27] C. Gentry, *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[28] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the Workshop on Selected Areas of Cryptography*, pp. 120–128, 1994.

[29] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," *Advances in Cryptology EUROCRYPT'98*, pp. 308–318, 1998.

[30] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology EUROCRYPT'99*, pp. 223–238, Springer, 1999.

[31] Y. Wang, X. Li, X. Li, and Y. Wang, "A survey of queries over uncertain data," *Knowledge and information systems*, vol. 37, no. 3, pp. 485–530, 2013.

[32] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 896–905, IEEE, 2007.

[33] Y. Tao, X. Xiao, and R. Cheng, "Range search on multidimensional uncertain data," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 3, p. 15, 2007.

[34] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 551–562, ACM, 2003.

[35] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei, "Probabilistic reverse nearest neighbor queries on uncertain data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 4, pp. 550–564, 2010.

[36] J. Li, B. Wang, and G. Wang, "Efficient probabilistic reverse k-nearest neighbors query processing on uncertain data," in *International Conference on Database Systems for Advanced Applications*, pp. 456–471, Springer, 2013.

[37] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[38] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pp. 24–24, IEEE, 2006.

[39] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 106–115, IEEE, 2007.

[40] J. J. Kim and W. E. Winkler, "Multiplicative noise for masking continuous data," in *Statistical Research Division, US Bureau of the Census, Washington DC*, Citeseer, 2003.

[41] K. Liu, C. Giannella, and H. Kargupta, "A survey of attack techniques on privacy-preserving data perturbation methods," *Privacy-Preserving Data Mining*, pp. 359–381, 2008.

[42] J. Vreeken, M. Van Leeuwen, and A. Siebes, "Preserving privacy through data generation," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pp. 685–690, IEEE, 2007.

[43] H. Roy, M. Kantarcioglu, and L. Sweeney, "Practical differentially private modeling of human movement data," in *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 170–178, Springer, 2016.

[44] D. F. Nettleton and J. Salas, "A data driven anonymization system for information rich online social network graphs," *Expert Systems with Applications*, vol. 55, pp. 87–105, 2016.

[45] Y. Xiao, L. Xiong, and C. Yuan, "Differentially private data release through multidimensional partitioning.," *Secure Data Management*, vol. 6358, pp. 150–168, 2010.

[46] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu, "Differentially private spatial decompositions," in *Data engineering (ICDE), 2012 IEEE 28th international conference on*, pp. 20–31, IEEE, 2012.

[47] W. Lu, G. Miklau, and V. Gupta, "Generating private synthetic databases for untrusted system evaluation," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 652–663, IEEE, 2014.

[48] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[49] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in *INFOCOM, 2014 Proceedings IEEE*, pp. 754–762, IEEE, 2014.

[50] K. Liu, "Paillier's cryptosystem in java." http://www.csee.umbc.edu/%7Ekunliu1/research/Paillier.html.

[51] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 129–142, 2015.

[52] J. M. Loh, "A valid and fast spatial bootstrap for correlation functions," *The Astrophysical Journal*, vol. 681, no. 1, p. 726, 2008.

[53] P. García-Soidán, R. Menezes, and Ó. Rubiños, "Bootstrap approaches for spatial data," *Stochastic environmental research and risk assessment*, vol. 28, no. 5, pp. 1207–1219, 2014.