Privacy Preserving Group Nearest Neighbor Queries

Tanzima Hashem
Department of CSSE
National ICT Australia (NICTA)
University of Melbourne
Victoria, Australia

Lars Kulik
Department of CSSE
National ICT Australia (NICTA)
University of Melbourne
Victoria, Australia

Rui Zhang
Department of CSSE
University of Melbourne
Victoria, Australia

{thashem,lars,rui}@csse.unimelb.edu.au

ABSTRACT

User privacy in location-based services has attracted great interest in the research community. We introduce a novel framework based on a decentralized architecture for privacy preserving group nearest neighbor queries. A group nearest neighbor (GNN) query returns the location of a meeting place that minimizes the aggregate distance from a spread out group of users; for example, a group of users can ask for a restaurant that minimizes the total travel distance from them. We identify the challenges in preserving user privacy for GNN queries and provide a comprehensive solution to this problem. In our approach, users provide their locations as regions instead of exact points to a location service provider (LSP) to preserve their privacy. The LSP returns a set of candidate answers that includes the actual group nearest neighbor. We develop a *private filter* that determines the actual group nearest neighbor from the retrieved candidate answers without revealing user locations to any involved party, including the LSP. We also propose an efficient algorithm to evaluate GNN queries with respect to the provided set of regions (the users' imprecise locations). An extensive experimental study shows the effectiveness of our proposed technique.

Keywords

Group nearest neighbor queries, location, privacy, private filter

1. INTRODUCTION

Location-based services (LBSs) have been originally tailored for requests of a single user, for example, asking for the closest gas station or the positions of traffic jams along a route. The advancement of LBSs has led to a new range of real-time services such as location-based social networking [1] (e.g., Loopt [2], Friend Finder [25]) that enable a group of users to be involved in a single location-based query, for example, a group of users may want to meet at a place that minimizes the total travel distance for them.

However, frequent and continuous access to these services exposes users to privacy risks: a location service provider (LSP) might be able to derive sensitive and private information about a user's health, habits, and preferences from the user's locations. For example, if a user requests a LBS from a health center then the user's health condition could be inferred. Due to an increasing awareness of privacy risks, users might refrain from accessing LBSs, which would hinder the proliferation of these services [3, 20].

Current research focuses on developing techniques that preserve user privacy during the access of LBSs. Although there is a range of privacy preserving techniques (e.g., [5, 14, 19, 27]) for answering a nearest neighbor (NN) query, processing a group nearest neighbor (GNN) query in a privacy preserving manner has not been explored. In a GNN query, a group of users provide their current locations, and the LSP returns the location (e.g., a meeting place) that minimizes an aggregate distance for the group. The aggregate distance could be the total distance of all group members or the maximum distance of any group member to the meeting location. This paper is the first work to address the problem of answering GNN queries while preserving user privacy for all members, which we call the *private GNN query*.

To preserve a user's privacy while accessing LBSs, a number of approaches have been proposed that provide an LSP with an imprecise instead of an exact location (e.g., [8, 12, 14]). In a privacy preserving or simply *private NN query* [9], the LSP returns a set of candidate NNs with respect to the imprecise location (typically a region). A user who knows her exact location can easily determine the actual NN from the returned candidate answer set.

In our proposed solution for a private GNN query, an LSP returns a set of candidate GNNs with respect to a set of regions. However, finding the actual GNN from the candidate answer set is more difficult than it is for a private NN query; a user who knows her exact location cannot determine the actual GNN from the returned candidate answer set because the actual GNN depends on the exact locations of all users involved in the GNN query. Not only the LSP, but even a group member may invade the privacy of other members. There are occasions where group members may wish to hide their current locations from other members for personal reasons. For example, a user who is at a job interview may wish to hide this location if a group of colleagues includes superiors. To ensure a user's privacy, no one should have access to locations of others. The key challenge in a private GNN query is to determine the actual GNN without enabling the LSP or other members to infer locations of users in the group. In this paper, we propose a system that allows users to request private GNN queries.

We will show in Section 5.1 that a straightforward technique to determine the actual GNN that does not share the users' actual locations with others, is prone to the so called *distance intersection attack*. In this technique, each user updates the distance for the retrieved candidate answers with respect to the user's actual location. However, from these updates it is possible to identify the distance of users to the candidate answers. The distance intersection attack uses the identified distances to the locations of the candidate answers to triangulate the user's location. We propose the *private filter* technique to address this attack. Our private filter technique passes the retrieved candidate GNN answers in an aggregated form to each user in the group. Based on each user's location, the answers are modified in such a way that no group member can derive other member locations but the actual GNN can still be computed.

The computation of the candidate answers for the private filter requires an algorithm for the LSP to evaluate the GNN query with respect to a set of regions (the users' imprecise locations). A straightforward application of algorithms for answering a point-based GNN query [18, 21, 22, 23] to a set of regions would have to consider every point configuration, where each configuration consists of one point location from each region. This would incur a high computational and I/O overhead. In this paper, we extend the existing GNN algorithm [23] for point locations to evaluate GNN queries with respect to a set of regions, which is an important part of our overall solution. Our proposed algorithm does not need a separate computation for each point set removing major overheads.

In this paper, we identify how the privacy of group members can be invaded for GNN queries and develop a novel approach to preserve their privacy. In summary, our contributions are:

- We propose a framework to preserve each group member's privacy during the access of LBSs as a group. The advantage of our framework is its decentralized architecture, which does not require any intermediary trusted server.
- We provide novel filtering techniques that find the actual GNN from a set of candidate GNNs without disclosing a member's location to others. Our techniques prevent the distance intersection attack.
- We extend the existing GNN algorithm [23] for point locations to regions, which is a necessary component to provide the candidate GNNs efficiently while preserving the privacy of group members.
- We evaluate our techniques in extensive experiments.

Section 2 presents the problem setup. In Section 3, we discuss related privacy approaches and GNN techniques. Section 4 presents our framework and overviews our system. The private filter techniques and the algorithm to evaluate a GNN query for a set of regions are described in Section 5 and 6, respectively. Section 7 reports experimental results and Section 8 concludes the paper.

2. PROBLEM SETUP

We assume a system architecture where the users and the LSP are connected through a network (e.g., the cellular network or the Internet). The problem of privacy preserving GNN queries (private GNN queries) is described as follows. Given a group of n users $u_1, u_2, \ldots, u_n$ located at points $l_1, l_2, \ldots, l_n$, respectively, issue a query for the group nearest data point (GNN). The formal definition of the GNN query is given below:

DEFINITION 2.1. ***(GNN Query).*** *Let D be a set of data points in a 2-dimensional space, Q be a set of n query points $\{q_1, q_2, \ldots, q_n\}$ and f be an aggregate function. The GNN query finds a data point p from D, such that for any $p' \in D - \{p\}$, $f(Q,p) \leq f(Q,p')$.*

In private GNN queries, the users in the group do not reveal their exact locations to the LSP; instead they provide regions $R_1, R_2, \ldots, R_n$ that contain $l_1, l_2, \ldots, l_n$, respectively. Thus the LSP needs to return a set of candidate data points A with respect to the provided regions that include the GNN for the actual user locations. Afterwards the actual GNN has to be computed from A without revealing the location of any user to any group member, not even in the imprecise format. In general, a group may be interested in finding k data points that have the k smallest aggregate distances, known as *k group nearest neighbor (kGNN) query*.

As we have seen in Section 1, the problem of privacy preserving kGNN queries has two parts: (i) The group of users use a technique, called *private filter*, to find the actual k GNNs from the retrieved set of candidate answers without compromising their privacy, and (ii) The LSP evaluates the kGNN query with respect to a set of regions to provide the candidate answers for the private filter while preserving user privacy. We formally define the private filter and the kGNN query w.r.t. regions as follows.

DEFINITION 2.2. ***(Private Filter).*** *Let A be a set of candidate data points, $\{u_1, u_2, \ldots, u_n\}$ be a group of n users, f be an aggregate function, and k be a positive integer. The precise location l_i of a user u_i is only known to u_i. A* private filter *is a mechanism that computes the k GNNs from A for the set $\{l_1, l_2, \ldots, l_n\}$ with respect to f without allowing others to identify any point location l_i.*

DEFINITION 2.3. ***(kGNN Query w.r.t. Regions).*** *Let D be a set of data points in a 2-dimensional space, $\{R_1, R_2, \ldots, R_n\}$ be a set of n query regions, r_i be any point in R_i for $1 \leq i \leq n$, and f be an aggregate function. The kGNN query w.r.t. regions returns the set of candidate data points A that includes all data points having the j^{th} smallest ($1 \leq j \leq k$) value for f with respect to every point set $\{r_1, r_2, \ldots, r_n\}$.*

In this paper, we focus on two aggregate functions SUM and MAX, which return the total distance and the maximum distance from the users to a data point, respectively.

3. RELATED WORK

Section 3.1 discusses state-of-the-art techniques for preserving user privacy in LBSs, and Section 3.2 reviews existing methods to evaluate GNN queries for a set of query points.

3.1 User Privacy in LBSs

Most research (e.g., [8, 12, 19]) to preserve user privacy in LBSs is based on a centralized architecture, where an intermediary trusted server acts as a privacy protector for the users. However, such a centralized architecture has a single point of failure, incurs bottlenecks due to communication overheads, and faces privacy threats as the intermediary server stores all information in a single place. Therefore, a few decentralized approaches (e.g., [7, 11, 14]) eliminate the role of an intermediary trusted server and preserve a user's privacy in cooperation with her peers. All approaches, centralized or decentralized are developed for a single user accessing LBSs.

Similarly for a private GNN query, in a centralized architecture each user in the group can send her exact location to the intermediary server, which forwards the GNN query with regions instead of the exact user locations to the LSP. The LSP returns the candidate answers with respect to the set of regions to the intermediary server. Since the intermediary server knows the exact locations of all users, it can compute the actual GNN and forward the actual answer to the group. To overcome the mentioned limitations of a centralized architecture, we propose a framework based on a decentralized architecture to access LBSs for a private GNN query.

Several techniques for hiding a user's location from the LSP without an intermediary server have been studied in the literature.

Most of these techniques (e.g., [7, 11, 10, 16, 14]) exploit P2P network (e.g., Bluetooth, WiFi), where an imprecise location of the user is computed as a rectangle or circle that includes $K-1$ other users' locations in addition to the location of the user requesting the query so that the user's location becomes K-anonymous. In [7, 11, 10], the users need to trust their peers with their locations in order to compute their imprecise locations.

In [16], a user who requires a service forms a group with $K-1$ other users through their proximity information and then the group progressively finds a bounding box as their rectangle that covers all users' locations. To compute the rectangle, in this technique the users do not need to share their actual locations with anyone, not even their peers. However, we do not use this technique to compute the user's rectangle to request a private kGNN query, because our proposed private filter requires that the user's rectangle sent to the LSP are not revealed to anyone, whereas in this technique a group of users needs to use the same rectangle (i.e., the user's rectangle is revealed to others).

In [14], each user computes her local imprecise location with a rectangle that includes her exact location, where the rectangle area is considered as privacy metric. When a user requires to access a LBS, she collects her peers' local imprecise locations and then computes her global imprecise location for the LSP as a minimum bounding rectangle that includes $K-1$ others' local imprecise locations (i.e., rectangles) and her exact location. Since neither the user's actual location nor the rectangle sent to the LSP are revealed to others, we use this technique to request a private GNN query.

Besides K-anonymity and imprecision, space transformation [17] and private information retrieval techniques [9] are also used to preserve the privacy of users. However, the architecture for both of these techniques require an encrypted database. In this paper, we assume that the users in a group disclose their imprecise locations to the LSP, which evaluates the queries based on these imprecise locations on a non-encrypted database.

We note that our system assumes that each user collaborates with others. If users are malicious, our approach can be complemented with secure multi-party protocols (e.g., [6]).

3.2 Group Nearest Neighbor Queries

kGNN queries are introduced by Papadias et al. [22]. kGNN queries are also known as aggregate nearest neighbor queries [23, 21]. In [22], the authors have developed three different methods, MQM (multiple query method), SPM (single point method) and MBM (minimum bounding method), to evaluate a GNN query that minimizes the total distance from a set of query points to a data point. In [23], Papadias et al. have extended these methods to minimize the minimum and maximum distance in addition to the total distance with respect to a set of query points. All these methods assume that the data points are indexed using an R-tree [13] and can be implemented using both *depth first search* (DFS) [24] and *best first search* (BFS) [15] algorithms.

The basic idea of MQM is to continue an incremental search for the nearest data point of each query point in the set and compute the aggregate distance from all query points for each retrieved data point. The search ends when it is ensured that the aggregate distance of any non-retrieved data point in the database is greater than the current k^{th} minimum aggregate distance, i.e, when the k GNNs are already found. The disadvantage of MQM is that it traverses the R-tree multiple times and may access the same data point more than once.

SPM and MBM, on the other hand, find the k GNNs in a single traversal of the R-tree. SPM approximates the centroid of the query distribution area and continues the search with respect to the centroid until the actual k GNNs are determined. MBM searches in order of the minimum aggregate distance of R-tree nodes from the set of query points. In SPM and MBM, the authors have proposed strategies to prune the R-tree nodes/data points while traversing the R-tree using the centroid and the minimum bounding box of the set of query points, respectively. They have also shown the conditions to terminate the search when k GNNs are found. Experimental results [22, 23] show that the performance of MBM is better than those of SPM and MQM as it traverses the R-tree once and takes the query distribution area into account.

In [18], Li et al. have approximated the query distribution area using an ellipse and used a distance or a minimum bounding box derived from the ellipse to prune the R-tree nodes/data points for processing kGNN queries. In [26], Luo et al. have proposed an algorithm to evaluate a GNN query only for non-indexed data points using projection-based pruning strategies and in [21], Namnandorj et al. have developed algorithms for both indexed and non-indexed data points by estimating a search space using a vector property.

This paper is the first study to propose an algorithm for processing a kGNN query with respect to a set of regions instead of a set of points in order to preserve user privacy.

4. FRAMEWORK BASED ON A DECENTRALIZED ARCHITECTURE

In this section, we first present a framework for processing a private GNN query based on a decentralized architecture, which eliminates the need for any intermediary trusted server. Then, we give an overview of our proposed system.

In our proposed framework a coordinator for the group is selected randomly before a query request. The coordinator assists in processing the private kGNN query and can be a group member or anyone outside the group who does not participate in the query. Note that the coordinator differs from an intermediary server in a centralized architecture because the coordinator can be different for every GNN query. Moreover, the coordinator only knows the user identities in the group and the type of query requested but has no knowledge about the user locations. The total process of accessing a private kGNN query is performed in three steps: (i) sending the query, (ii) evaluating the query, and (iii) finding the answer. We detail these components in the following subsections.

4.1 Sending the Query

Each user in the group first registers to the coordinator with their identities (e.g., IP address, phone number) and receives a query identity (QID) from the coordinator. Each group user sends her imprecise location and the QID anonymously to the LSP using either a pseudonym service [11] in the Internet or through a randomly selected peer [14, 7] connected in a wireless personal area network (e.g., Bluetooth or 802.11). These techniques hide the users' identities from the LSP as well as from the cellular infrastructure provider. The coordinator only sends the kGNN query for the required service, which includes the QID, the description of the required service, the value for k and the number of users in the group to the LSP.

4.2 Evaluating the Query

After receiving the request, the LSP evaluates the kGNN query with respect to the set of regions. Since the LSP does not know the exact user locations, it cannot determine the actual k GNNs. Therefore, it returns a set of candidate answers that include the actual GNNs to the coordinator.

4.3 Finding the Answer

The final step is to determine the actual k GNNs without revealing the user locations to anyone. The retrieved answer set has to go through all users of the group. Each user updates the distance to the candidate GNN answers with respect to her actual location. The communication between the users in the group can be done with or without the coordinator.

In the first case (with the coordinator), the coordinator randomly selects one of the user identities in the group and sends the answer set to that user. After receiving the modified answer set, the coordinator marks that user's identity as *visited*. The coordinator repeats this procedure with the remaining unmarked user identities.

In the second case (without the coordinator), the coordinator forwards the retrieved answer set together with the list of identities of all participants to a randomly selected user in the group. The selected user modifies the candidate answers and marks her identity as visited. Then she randomly selects a user with an unmarked identity and forwards the updated answer set and the list of identities to the next selected user.

After the answer set has been modified by all users, the coordinator (in the first case) or the last selected user (in the second case) sends the actual GNNs to all users in the group.

4.4 Overview of Our System

We propose a system for processing privacy preserving kGNN queries based on the framework above. We assume that users in the group compute their imprecise locations as rectangles using the method in [14] as discussed in Section 3.1. We present an algorithm to evaluate the kGNN query with respect to the set of rectangles and develop techniques for a private filter that finds the actual k GNNs for the aggregate functions SUM and MAX.

For ease of understanding, we first discuss the private filter techniques in Section 5. We show a straightforward method to determine the actual GNNs from the retrieved answer set where the users do not disclose their locations to anyone; instead they update the distance to the candidate GNNs using their actual distances. However, these updates enable others to use the received distance updates to the candidate GNNs and compute a user's actual location using 2-dimensional (2D) trilateration. We call this attack a *distance intersection attack* on a user's privacy. We propose private filter techniques where the distance of candidate answers are updated by each user in such a way that no party can identify an individual's distance from the candidate answers and thus cannot apply a distance intersection attack to determine a user's location.

Then, we present an algorithm to evaluate a kGNN query with respect to a set of rectangles in Section 6. Since, our algorithm deals with a set of query rectangles instead of query points, the measured distance from query rectangles is a range instead of a fixed value and the search for GNNs is made based on that range. Our algorithm finds the GNNs in a single search on the database whereas existing algorithms for a set of query points require multiple searches for finding the GNNs with respect to a set of query rectangles.

5. PRIVATE FILTERS

5.1 Minimizing the Total Distance

Without loss of generality, consider an example scenario for a private kGNN query with a group of five users and $k = 2$. The users $\{u_1, u_2, \ldots, u_5\}$ provide their query rectangles $\{R_1, R_2, \ldots, R_5\}$, respectively, to an LSP. The LSP returns the locations of a set of data points A: $\{p_1, p_2, \ldots, p_8\}$ that includes the 2 GNNs with respect to the actual locations $\{l_1, l_2, \ldots, l_5\}$ of the users. The locations of data points in A, the actual and imprecise locations of the users are shown in Figure 1, and the actual distances of the users to all data points in A are presented in Table 1.

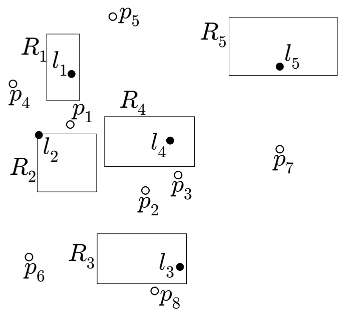


Figure 1: An example scenario.

First, we show a straightforward technique to determine the actual GNNs and the privacy attack associated with this technique. As mentioned in Section 4, A has to be updated by all users in the group with respect to their exact locations for finding the actual GNNs from A. Suppose user u_1 first receives A from the coordinator c. Then u_1 updates A: $\{p_1, p_2, \ldots, p_8\}$ by inserting a new distance field for each data point in A and initializing the fields with her actual distances from those data points as A:$\{(p_1, 3), (p_2, 8.5) \ldots, (p_8, 14)\}$. Then, A is forwarded to a randomly selected user u_2, either directly or via c. The user u_2 adds her actual distances for all data points in A with those of u_1 and forwards them to another user. This process continues until all users have added their actual distances for all data points in A. After all updates, the final value of the distance field for each data point in A represents the total distance of that data point to all group members (see Table 2). Thus, the last user (u_5 in this example) or the coordinator c can determine the 1^{st} and 2^{nd} group nearest data points p_3 and p_1, and sends them to all participant users in the group.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
u_1	3	8.5	9	3.5	4.5	11.5	13.5	14
u_2	2	7.5	8.5	3.5	8.5	7.5	14.5	12
u_3	11	5	5.5	15	15.5	9	9	2
u_4	6	3.5	2	10	8.5	11	6.5	9
u_5	12.5	10.5	8.5	15.5	10	18.5	5	15.5

Table 1: Actual distance from the users to the data points in *A*

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
u_1	3	8.5	9	3.5	4.5	11.5	13.5	14
u_2	5	16	17.5	7	13	19	28	26
u_3	16	21	23	22	28.5	28	37	28
u_4	22	24.5	25	32	37	39	43.5	37
u_5	34.5	35	33.5	47.5	47	57.5	48.5	52.5

Table 2: Updated distances after adding each user's actual distance to the data points in *A*

However, the privacy of users can be violated in this technique using the *distance intersection attack*. The distance intersection attack is based on 2D trilateration. If a user's distance from a known location is revealed, then the user's location has to be on the circle centered at the known location with the radius of the revealed distance. If a user's distances from two known locations are revealed, the user's location is one of the two intersections of the circles. If a user's distances from three or more known locations are revealed, the user's exact location is the intersection point of all circles.

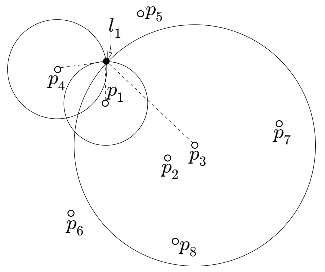


Figure 2: An example of distance intersection attack.

In our example, if u_2 receives a message from u_1 directly, the message includes A, the identities of $\{u_1,u_2,u_3,u_4,u_5\}$, and the identity of u_1 marked as visited. Inspecting the visited field, u_2 knows that she is the second randomly selected user who receives A. Since u_2 also knows that the distances in A are the actual distances of u_1 to $\{p_1,p_2,\ldots,p_8\}$, the unknown location l_1 of u_1 can be computed from any of the three revealed distances using the distance intersection attack (Figure 2). In the case that the communication among the group members is done via a coordinator to hide their identities from each other, u_2 can again determine a location from the intersection point of the circles. However, u_2 does not know which user is located at that intersection point, because u_2 has no access to the list of identities showing that only the identity u_1 is marked as visited. In this case, the coordinator c can compute the exact locations of all users using the distance intersection attack. The coordinator c monitors A before sending it to u_i and after receiving it from u_i and then computes the actual distances of u_i for all data points in A. For example, the actual distance of u_4 to p_4 is found by deducting 22 (observed before sending A to u_4) from 32 (observed after receiving A from u_4) in Table 2.

We present now our private filter technique that counters the distance intersection attack on the users' privacy. Let n be the number of users in the group, where $n > 2$, and $MaxDist(R_i,p_h)$ be a function that returns the maximum Euclidean distance between a user's rectangle R_i and a data point p_h for a positive integer h. In our private filter technique, the LSP returns for each data point $p_h \in A$ the sum of the maximum distances of p_h to the query rectangles $d_{max}(p_h)$, expressed as:

$$d_{max}(p_h) = \textstyle\sum_{i=1}^{n} MaxDist(R_i,p_h) \qquad (1)$$

On receiving A, a user u_i in the group updates d_{max} for all data points with respect to her actual position l_i. Let the function $Dist(l_i,p_h)$ return the Euclidean distance between a user's actual location l_i and p_h. The user u_i computes $d'_{max}(p_h)$ for a data point p_h using the following equation:

$$d'_{max}(p_h) = d_{max}(p_h) - MaxDist(R_i,p_h) + Dist(l_i,p_h) \qquad (2)$$

Then u_i updates $d_{max}(p_h)$ by assigning $d'_{max}(p_h)$ to $d_{max}(p_h)$ for a data point p_h. After completing the updates for all data points, u_i forwards A to another user, either directly or via the coordinator. Each user updates d_{max} for all data points in A using this procedure.

Let X represent a subset of users in the group who have already updated d_{max} for all data points in A and Y represent the remaining users in the group who have not yet received and updated A. In every step of the private filter technique, $d_{max}(p_h)$ can be in general expressed by the following equation:

$$d_{max}(p_h) = \sum_{u_x \in X} Dist(l_x,p_h) + \sum_{u_y \in Y} MaxDist(R_y,p_h) \qquad (3)$$

As a result, when $d_{max}(p_h)$ of a data point p_h has been updated with respect to all users' exact locations, it represents the aggregate distance ($\sum_{i=1}^{n} Dist(l_i,p_h)$) from p_h to the group. Table 3 shows the steps for updating d_{max} by every user for the given example. After the updates of u_5, $d_{max}(p_1), d_{max}(p_2), \ldots, d_{max}(p_8)$ represent the actual aggregate distance of $\{p_1,p_2,\ldots,p_8\}$ from the group of users $\{u_1,u_2,\ldots,u_5\}$ (see the last row of Table 3). Depending on the communication method used, u_5 or the coordinator c forwards 2 GNNs, p_3 and p_1, to all users in the group.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
LSP	46.5	46	44	60.5	56.5	68.5	62	63
u_1	44	43.5	41	60	54.5	66	60	60
u_2	42	43.5	41	55	51.5	65	60	60
u_3	41	42	40	54	51	64.5	55.5	58.5
u_4	39.5	40	38.5	52	50.5	62.5	51.5	57
u_5	34.5	35	33.5	47.5	47	57.5	48.5	52.5

Table 3: Updated $d_{max}(p_h)$ with respect to each user's actual distance from data points in A

In this technique, the privacy of all users is preserved in both scenarios: without or with the coordinator c. In the first scenario, the second randomly selected user u_2 cannot compute the actual distances of the first randomly selected user u_1 for data points in A as the actual distance of u_i from the data points are hidden in the revealed d_{max}s as shown in Equation 3. In the second scenario, although c can monitor the change in d_{max}s for data points in A before sending it to u_i and after receiving it from u_i, c cannot determine the actual distance of u_i from any data point in A because the coordinator does not know the locations of the users' rectangles. For example, c monitors the change of $d_{max}(p_4)$ from 54 to 52 before sending A to u_4 and after receiving A from u_4, respectively (see Table 3). However, as the location of R_4 is unknown to c, c cannot determine $MaxDist(R_4,p_4)$ to compute $Dist(l_4,p_4)$. Note that even if the coordinator colludes with the LSP neither the LSP nor the coordinator can find the one to one mapping between the sets of users' rectangles and identities (i.e., which rectangle belongs to which user). Knowing only the set of rectangles does not allow the coordinator to compute a $Dist(l_4,p_4)$.

The private filter technique discussed so far cannot perform any pruning of data points from A until all users in the group update A with respect to their actual locations. We call this private filter for SUM a *final pruning private filter* (SUM_FPPF). In the next step, we propose an *incremental pruning private filter* for SUM (SUM_IPPF) that allows each user to perform a local pruning of those data points from the answer set that cannot be the actual GNNs.

In SUM_IPPF, the LSP provides the sum of the minimum distances from query rectangles d_{min} in addition to the sum of the maximum distances from query rectangles d_{max} for all data points in A. The addition of d_{min} allows a user to perform a local pruning of the data points from A after the update and to send a smaller answer set to the next user. Let $MinDist(R_i,p_h)$ be a function that returns the maximum Euclidean distance between R_i and p_h. The LSP computes $d_{min}(p_h)$ as follows:

$$d_{min}(p_h) = \sum_{i=1}^{n} MinDist(R_i,p_h) \qquad (4)$$

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
LSP	21, 46.5	23, 46	23.5, 44	35, 60.5	35, 56.5	40, 68.5	40.5, 62	41.5, 63
u_1	22.5, 44	24.5, 43.5	25, 41	36.5, 60	37.5, 54.5	43, 66	41.5, 60	43, 60
u_2	24, 42	29, 43.5	28.5, 41	36.5, 55	39, 51.5	46.5, 65	45, 60	48, 60
u_3	28, 41	31.5, 42	30.5, 40	41, 54	41.5, 51	X	X	X
u_4	32, 39.5	33.5, 40	32, 38.5	45, 52	X	X	X	X
u_5	34.5, 34.5	35, 35	33.5, 33.5	X	X	X	X	X

Table 4: Updated $d_{min}(p_h)$ and $d_{max}(p_h)$ with respect to each user's actual distance from data points in A

On receiving A, each user updates both d_{min} and d_{max} for all data points in A. Similar to $d'_{max}(p_h)$, a user computes $d'_{min}(p_h)$ for a data point p_h using the following equation:

$$d'_{min}(p_h) = d_{min}(p_h) - MinDist(R_i, p_h) + Dist(l_i, p_h) \qquad (5)$$

Afterwards the user updates $d_{min}(p_h)$ by assigning $d'_{min}(p_h)$ to $d_{min}(p_h)$.

Algorithm 1: SUM_IPPF(R_i, l_i, k, A)

```
Input  : The user's rectangle $R_i$ and exact point location $l_i$, the
         number of required data points $k$, and the answer set
         $A := \cup_h\{p_h, d_{min}(p_h), d_{max}(p_h)\}$
Output: Updated answer set $A$.
1.1 for each $p_h \in A$ do
1.2     compute $d'_{min}(p_h)$ using Equation 2
1.3     $d_{min}(p_h) \leftarrow d'_{min}(p_h)$
1.4     compute $d'_{max}(p_h)$ using Equation 5
1.5     $d_{max}(p_h) \leftarrow d'_{max}(p_h)$
1.6 $maxdist_k \longleftarrow kMin(\cup_h\{d_{max}(p_h)\})$
1.7 for each $p_h \in A$ do
1.8     if $d_{min}(p_h) > maxdist_k$ then
1.9         remove $\{p_h, d_{min}(p_h), d_{max}(p_h)\}$ from $A$
```

Algorithm 1 summarizes the steps performed by a user on receiving A for the aggregate function SUM. After updating d_{min} and d_{max} for all data points in A, the user finds the k^{th} smallest of all d_{max} as $maxdist_k$ using the function $kMin$. Then d_{min} of every data point in A is compared with $maxdist_k$. If $d_{min}(p_h)$ of a data point p_h is greater than $maxdist_k$, then p_h is removed from A as p_h can never be one of the k nearest data point from the group. Table 4 shows the steps for updating d_{min} and d_{max}, and the pruning of data points by every user in our example. From Table 4, we see that the user u_2 determines $maxdist_k$ as 42 for $k = 2$ and removes p_6 ($d_{min}(p_6) = 46.5$), p_7 ($d_{min}(p_7) = 45$), and p_8 ($d_{min}(p_8) = 48$) from A. Hence, the next user u_3 can process a smaller answer set, and more importantly, the local pruning reduces the communication overhead among the users.

For SUM_IPPF, a special case may arise if a data point p_h overlaps with all rectangles $\{R_1, R_2, \ldots, R_n\}$. In this case, the retrieved $d_{min}(p_h)$ (i.e., $\sum_{i=1}^{n} MinDist(R_i, p_h)$) from the LSP is 0 and if the users communicate via the coordinator, the coordinator learns each user's distances to p_h. Therefore if any $d_{min}(p_h)$ is 0, users communicate directly to avoid the distance intersection attack.

In summary, for any group size $n > 2$, the discussed private filter techniques find the actual GNNs without revealing users' locations to others. However for a group of two users (i.e., $n = 2$), an extra attention is required: if users communicate directly for $n = 2$, a user u_2 determines herself as the second user by observing the list of identities with one identity marked as visited. Then for every data point p_h in the answer set, u_2 can determine $Dist(l_1, p_h)$ by subtracting $MaxDist(R_2, p_h)$ from $d_{max}(p_h)$ as shown in Equation 3. Therefore, u_2 can apply the distance intersection attack to find u_1's precise location l_1. On the other hand, if the second user u_2 receives the candidate data points from the coordinator then she does not know that she is the second user as she does not have the list of identities with one identity marked as visited. Thus, $(MaxDist(R_1, p_h) + MaxDist(R_2, p_h))$ or $(Dist(l_1, p_h) + MaxDist(R_2, p_h))$ could be $d_{max}(p_h)$, and user u_2 cannot discover $Dist(l_1, p_h)$. Hence for $n = 2$, users need to communicate via the coordinator to find the actual GNNs.

The following theorem shows the correctness of our proposed private filter SUM_FPPF.

THEOREM 5.1. *The private filter* SUM_FPPF *prevents the distance intersection attack on a user's location.*

PROOF. We know that in order to apply the distance intersection attack for finding a user's actual location, the coordinator or other users involved in the private filter need to know the distance of that user to the data points in the answer set. It is not possible to determine a user u_i's $Dist(l_i, p_h)$ by the coordinator or any other user u_j for $i \neq j$ from Equations 2 and 3, if there is an unknown variable. For any group size, since the coordinator does not know u_i's R_i, it cannot determine $Dist(l_i, p_h)$ using Equation 2 after $d_{max}(p_h)$ has been updated by u_i. On the other hand for $n > 2$, on receiving $d_{max}(p_h)$, u_j cannot determine $Dist(l_i, p_h)$ using Equation 3 because u_j does not know others' l_xs and R_ys, where $x \neq i$ and $y \neq j$. For $n = 2$ since users communicate via the coordinator, u_j does not have the list of identities and thus cannot know whether u_i is in X or Y in Equation 3, which prevents u_j to determine $Dist(l_i, p_h)$. □

Similarly, we can prove the correctness of SUM_IPPF.

5.2 Minimizing the Maximum Distance

In this section, we consider private kGNN queries that minimize the *maximum* distance of a group of users from the data points. Similar to the case of minimizing the total distance, we cannot use the straightforward technique due to its vulnerability of the distance intersection attack.

We can use both techniques, FPPF and IPPF proposed in Section 5.1, with some modifications for finding the data point that has the minimum maximum distance from the group of users. In this case, the LSP uses the aggregate function MAX instead of SUM to compute $d_{min}(p_h)$ and $d_{max}(p_h)$ for a data point p_h as shown in the following two equations:

$$d_{min}(p_h) = \max_{i=1}^{n} MinDist(R_i, p_h) \qquad (6)$$

$$d_{max}(p_h) = \max_{i=1}^{n} MaxDist(R_i, p_h) \qquad (7)$$

Algorithm 2 shows the steps of MAX_IPPF. In case of the aggregate function MAX, a user u_i only updates $d_{min}(p_h)$ as $Dist(l_i, p_h)$ when $Dist(l_i, p_h)$ is larger than the current $d_{min}(p_h)$ for a data point p_h (Lines 2.2-2.3). By construction, there is always at least a user in the group whose distance from p_h is equal to or greater than $d_{min}(p_h)$.

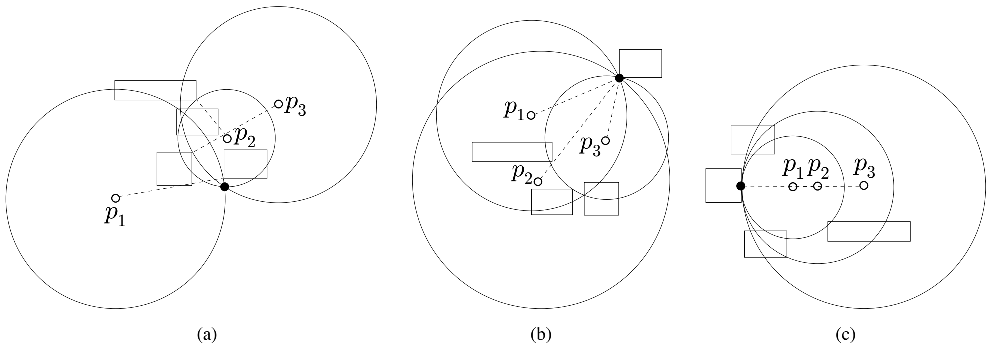


Figure 3: Examples scenarios of circles with radii equal to d_{min}s retrieved from the LSP

Algorithm 2: MAX_IPPF($R_i, l_i, A, maxdist_k$)

```
Input  : The user's rectangle $R_i$ and exact point location $l_i$, the
         answer set $A := \cup_h\{p_h, d_{min}(p_h)\}$, and $maxdist_k$
Output: Updated answer set $A$.
2.1 for each $p_h \in A$ do
2.2     if $Dist(l_i, p_h) > d_{min}(p_h)$ then
2.3         $d_{min}(p_h) \longleftarrow Dist(l_i, p_h)$
2.4 for each $p_h \in A$ do
2.5     if $d_{min}(p_h) > maxdist_k$ then
2.6         remove $\{p_h, d_{min}(p_h)\}$ from $A$
```

On the other hand, a user cannot modify $d_{max}(p_h)$ even if $Dist(l_i, p_h)$ is smaller than the current $d_{max}(p_h)$ as the other users in the group can also have distances from p_h equal to $d_{max}(p_h)$. Thus, in contrast to Algorithm 1, $maxdist_k$ never needs to be updated and remains constant. Therefore, the LSP computes $maxdist_k$ from d_{max} of the data points in A and directly sends $maxdist_k$ instead of sending d_{max} for each data point.

If a user updates $d_{min}(p_h)$ in Algorithm 2, it represents the user's actual distance from p_h (Line 2.3). If the communication among the users is done via a coordinator c for filtering the retrieved answer set from the LSP, c can observe d_{min} for each data point in A before sending A to a user u_i and after receiving it back from u_i, and determine which d_{min}s have been updated by u_i. The changed $d_{min}(p_h)$ denotes the actual distance of u_i from p_h. Thus c can compute the location of u_i with the distance intersection attack using d_{min}s changed by u_i. Thus in our proposed private filter for the aggregate function MAX, users avoid the coordinator and communicate directly.

In the direct communication, after performing the update a user sends A directly to another user in the group whose identity has not been yet marked as visited. In order to apply the distance intersection attack for revealing a user's unknown location, we need to know the user's distances from known locations. A user who receives A knows the location of data points in A and the distances in the form of d_{min} for each data point in A. The user also knows that a $d_{min}(p_h)$ represents either the distance of p_h from a user's actual location or the distance of p_h returned by the LSP. However, the user does not know which $d_{min}(p_h)$ corresponds to which user's actual distance from p_h as she has no knowledge about the previous states of A and the order in which the identities are marked as visited. Thus, the user who receives A cannot discover others' locations in the group using the distance intersection attack.

We know that the second user who receives A can easily identify the user who has received A before her by inspecting the visited field. We also know that if a subset of d_{min}s are the actual distances from the same user, then the circles with radii equal to those d_{min}s and centers at the corresponding locations of data points must intersect at a single point. Using these observations, one may argue that if the second user finds from the received d_{min}s that a number of circles intersect at a single point, then she would be able to identify the intersection point as the location of the first user. However, it is not guaranteed that the intersection point is the location of the first user, because the values of d_{min}s that are assigned by the LSP may have caused the intersection point and they might not have been changed by the first user at all.

Figure 3 shows some examples, where d_{min}s computed by the LSP are shown with dashed lines and the intersection point of all circles are shown with a black dot. In Figure 3(a), the intersection point of all circles does not refer to any user's location, and in Figure 3(b) and (c), the intersection point is the location of a user's provided rectangle which is not ensured to be the actual location of any user in the group as a user's actual location can be anywhere within the rectangle.

In contrast to SUM_FPPF, for MAX_FPPF, the LSP returns d_{min} instead of d_{max} for each data point in A. This is because if the LSP returns d_{max} for data points in A and a user u_i updates $d_{max}(p_h)$ as $Dist(l_i, p_h)$ if $Dist(l_i, p_h) < d_{max}(p_h)$, then after the update of A by the first user u_1, each d_{max} represents $Dist(l_1, p_h)$. As a result, the user who receives A as a second user can determine u_1's precise location l_1 using the distance intersection attack. In MAX_FPPF, the users update d_{min} for each data point in A as shown in Lines 2.2-2.3 of Algorithm 2 and determine the actual GNN after A has been updated by all users in the group.

There is a limitation of our proposed private filter techniques for the aggregate function MAX. It does not work for $n = 2$, which we leave for further investigation in our future research. For $n = 2$, after the update by the first user u_1 each $d_{min}(p_h)$ represents either $Dist(l_1, p_h)$ or $MinDist(R_2, p_h)$, where R_2 is the rectangle of the second user u_2. When u_2 receives A she can determine that she is the second user by observing the visited field in the list of identities and determine whether $d_{min}(p_h)$ represents $Dist(l_1, p_h)$ as she knows $MinDist(R_2, p_h)$. This allows u_2 to apply the distance intersection attack if $d_{min}(p_h)$ has been modified by u_1.

From the above discussion, we summarize that FPPF and IPPF enable users to request kGNN queries without revealing their locations to anyone with any group size for SUM and with a group size greater than two for MAX.

6. kGNN QUERIES W.R.T. REGIONS

In this section, we propose an algorithm for the LSP to process kGNN queries with respect to a set of rectangles (i.e, regions). Our algorithm uses a modified best first search (BFS) to find the candidate answers that include the k GNNs for any position of the users in their provided rectangles. We assume that the data points are indexed using an R^*-tree [4] in the database. Since the query is based on a set of rectangles instead of a set of points, the distance between a data point or an R^*-tree node and a query rectangle is defined with a range bounded by the minimum and maximum values.

We summarize the notation used in this section as follows:

- M: the minimum bounding box that encloses the given set of n query rectangles $\{R_1, R_2, \dots, R_n\}$.
- $MinDist(q,p)$ ($MaxDist(q,p)$): the minimum (maximum) Euclidean distance between q and p, where q represents R_i or M and p represents a data point or a minimum bounding rectangle of an R^*-tree node.
- $d_{min}(p)$ ($d_{max}(p)$): the aggregate distance (i.e., the total or maximum distance) of p computed from the minimum (maximum) distances between p and all query rectangles, where p again represents a data point or a minimum bounding rectangle of an R^*-tree node.
- $maxdist[k]$: the k^{th} smallest distance of already computed $d_{max}(p)$s.

The basic idea of our proposed algorithm is as follows. The algorithm starts the search from the root of the R^*-tree and inserts the root together with its $d_{min}(root)$ and $d_{max}(root)$ into a priority queue Q_p, where $d_{min}(root) = 0$ and $d_{max}(root) = f_{i=1}^{n}(MaxDist(R_i, root))$, f being SUM or MAX. The elements of Q_p are stored in order of their minimum d_{min}. Then the algorithm removes an element p from Q_p and checks whether p is an R^*-tree node or a data point. If p represents an R^*-tree node, then it retrieves its child nodes and enqueues them into Q_p if they might contain one of the candidate answers with respect to the set of rectangles. On the other hand, if p is a data point it is added to A until all data points have been found that are candidates for one of the k GNNs with respect to the set of rectangles. Algorithm 3 shows the steps of REGION_kGNN for evaluating kGNN queries with respect to a set of rectangular regions.

In the case of kGNN queries for a set of points, the algorithm terminates as soon as k data points have been dequeued from Q_p. However, for a set of rectangles the termination is not as simple, because the total or maximum distance of a data point from the query rectangles is a range $[d_{min}, d_{max}]$ instead of a fixed value. We know that the elements removed from Q_p are in order of minimum d_{min}, but we also need to maintain the order of already computed d_{max}s to check the termination condition of the algorithm. For this purpose an array $maxdist$ with k entries is maintained and initialized to ∞ (Line 3.3). The array $maxdist$ is sorted in order of minimum d_{max}s found so far. Each time p is inserted to Q_p, $maxdist$ is updated with respect to $d_{max}(p)$ (Line 3.22). The following heuristic describes the termination condition of the algorithm as no other data point can further qualify as a candidate answer once the condition is true.

HEURISTIC 6.1. *Let p be a data point or an R^*-tree node dequeued from Q_p. The algorithm terminates if $d_{min}(p) > maxdist[k]$.*

In REGION_kGNN, we use the variable *end*, initialized to 0, to terminate the algorithm. When the condition of Heuristic 6.1 is satisfied, *end* becomes 1 (Lines 3.7-3.8) and the algorithm terminates (Line 3.5). Figure 4 shows an intermediate state of running REGION_kGNN with $k = 2$ and $f =$ MAX, where the current A includes $\{p_2, 9, 15.7\}$, $\{p_3, 9, 13.5\}$, $\{p_1, 10, 17.5\}$, $\{p_7, 13.5, 15.5\}$, $\{p_4, 13, 20\}$, $\{p_5, 13, 16\}$, and $\{p_{10}, 14, 20.5\}$. Hence, at this stage

Algorithm 3: REGION_kGNN($R_1, R_2, \dots, R_n, k, f$)

```
Input : A set of rectangles $\{R_1, R_2, \dots, R_n\}$, the number of
        required data points $k$, and an aggregate function $f$ (SUM
        or MAX).
Output: $A$, a set of data points with their $d_{min}$ and $d_{max}$.
3.1  $A \leftarrow \emptyset$
3.2  $end \leftarrow 0$
3.3  $maxdist[1..k] \leftarrow \{\infty\}$
3.4  $Enqueue(Q_p, root, 0, f_{i=1}^{n}(MaxDist(R_i, root))$
3.5  while $Q_p$ is not empty and $end = 0$ do
3.6      $\{p, d_{min}(p), d_{max}(p)\} \leftarrow Dequeue(Q_p)$
3.7      if $d_{min}(p) > maxdist[k]$ then
3.8          $end \leftarrow 1$
3.9      else if $p$ is a data point then
3.10         $A \leftarrow A \cup \{p, d_{min}(p), d_{max}(p)\}$
3.11     else
3.12         for each child node $p_c$ of $p$ do
3.13             if $f =$ SUM then
3.14                 $d(p_c) \leftarrow n \times MinDist(M, p_c)$
3.15             else
3.16                 $d(p_c) \leftarrow MinDist(M, p_c)$
3.17             if $d(p_c) \leq maxdist[k]$ then
3.18                 $d_{min}(p_c) \leftarrow f_{i=1}^{n} MinDist(R_i, p_c)$
3.19                 if $d_{min}(p_c) \leq maxdist[k]$ then
3.20                     $d_{max}(p_c) \leftarrow f_{i=1}^{n} MaxDist(R_i, p_c)$
3.21                     $Enqueue(Q_p, p_c, d_{min}(p_c), d_{max}(p_c))$
3.22                     $Update(maxdist, d_{max}(p_c))$
3.23 return $A$;
```

$maxdist[1] = d_{max}(p_3) = 13.5$ and $maxdist[2] = d_{max}(p_7) = 15.5$. Let the next three elements in Q_p be $\{p_8, 14, 20\}$, $\{p_6, 16.5, 23.5\}$, and $\{p_9, 17.5, 21.5\}$. When $\{p_8, 14, 20\}$ is dequeued from Q_p, then $d_{min}(p_8) < maxdist[2]$. Therefore $\{p_8, 14, 20\}$ is added to A and $maxdist$ remains unchanged as $d_{max}(p_8)$ is greater than both $maxdist[1]$ and $maxdist[2]$. Next, $\{p_6, 16.5, 23.5\}$ is dequeued from Q_p and as $d_{min}(p_6) > maxdist[2]$, the algorithm terminates.

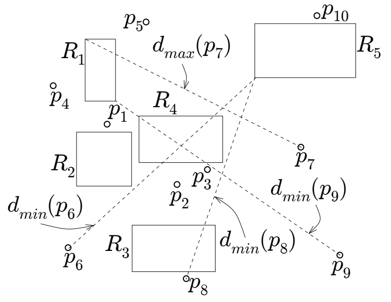


Figure 4: An intermediate state of running REGION_kGNN.

Note that not all visited data points or R^*-tree nodes are inserted to Q_p. Before inserting p into Q_p, the algorithm checks if p can be pruned with respect to the current $maxdist[k]$. As d_{min} and d_{max} involve a large number of distance computations, similar to [22, 23], REGION_kGNN tests in Line 3.17 if p can be pruned according to the following heuristic.

HEURISTIC 6.2. *A data point or an R^*-tree node p can be pruned if $n \times MinDist(M,p) > maxdist[k]$ for $f =$ SUM and if $MinDist(M,p) > maxdist[k]$ for $f =$ MAX.*

If p is not pruned using Heuristic 6.2, then Algorithm 3 uses the tighter condition $d_{min}(p) > maxdist[k]$ of Heuristic 6.1 to check if p can be discarded before inserting it into Q_p as shown in Line 3.19. Since $n \times MinDist(M,p) \leq d_{min}(p)$ for $f = $ SUM and $MinDist(M,p) \leq d_{min}(p)$ for $f = $ MAX, it may happen that p is not pruned using the condition of Heuristic 6.2 but satisfies the condition of Heuristic 6.1 and is pruned.

Note that for SUM the LSP directly returns A, a set of candidate answers with their d_{min}s and d_{max}s, to the coordinator. On the other hand, for MAX the LSP removes d_{max} of each data point from A, and returns $maxdist[k]$ and A that includes a set of candidate answers with their d_{min}s to the coordinator.

The following theorem proves the correctness of algorithm REGION_kGNN.

THEOREM 6.1. *If k is the number of required data points for a kGNN query with respect to a set of n query rectangles $\{R_1, R_2, \ldots, R_n\}$ with $r_i \in R_i$ for $1 \leq i \leq n$, then A includes all data points that have the j^{th} smallest ($1 \leq j \leq k$) value for f (SUM or MAX) with respect to every point set $\{r_1, r_2, \ldots, r_n\}$.*

PROOF. (By contradiction) Assume that p' is a data point that is not in A but has the j^{th} minimum value ($1 \leq j \leq k$) for f (SUM or MAX) with respect to a group of n points $\{r_1', r_2', \ldots, r_n'\}$, where each point r_i' can be located at any position in R_i. There can be two cases for $p' \notin A$: (i) the algorithm has terminated before p' is included in A, or (ii) p' or the R^*-tree node containing p' has been pruned.

We know that the aggregate distance of a data point p from $\{r_1, r_2, \ldots, r_n\}$ is within $d_{min}(p)$ and $d_{max}(p)$. $maxdist[k]$ represents the current k^{th} smallest d_{max}, and $maxdist[k]$ remains same or becomes smaller during the execution of the algorithm, because d_{max} of a R^*-tree node is greater or equal than those of its child nodes. According to our assumption, if p' is one of the k GNNs, then $d_{min}(p') \leq maxdist[k]$.

We consider first case (i). The algorithm terminates when $d_{min}(p) > maxdist[k]$, for any p dequeued from Q_p and Q_p is ordered by minimum $d_{min}(p)$. As p' has not been dequeued before p, i.e., $d_{min}(p') > d_{min}(p)$, which in turn means $d_{min}(p') > maxdist[k]$. Therefore, the first case for $p' \notin A$ does not apply as there are already k group nearest data points for $\{r_1', r_2', \ldots, r_n'\}$, whose d_{max}s are less than $d_{min}(p')$.

Let us assume case (ii) that p' is not included in A because it has been pruned before inserting into Q_p. However, p' is only pruned if it satisfies the condition of Heuristic 6.1 or Heuristic 6.2, which again means that $d_{min}(p') > maxdist[k]$ and contradicts our assumption that p' is one of the k GNN for $\{r_1', r_2', \ldots, r_n'\}$. $\square$

Although we present our algorithm for a set of query rectangles, our algorithm can evaluate kGNN queries for a set of query regions with any geometric shape.

7. EXPERIMENTS

In this section, we evaluate the performance of our proposed algorithms through extensive experiments. We vary the group size, the area of the minimum bounding box M that encloses the set of query rectangles, the area of a query rectangle, the number of required data points k, and the data set size in different sets of experiments. We use both real and synthetic data sets in our experiments. The data space is normalized into a span of $10,000 \times 10,000$ square units. The real data set C contains 62,556 postal addresses from California. We generate synthetic data sets U and Z using a uniform and a Zipfian distribution, respectively, and we vary the size of U and Z as 5000, 10,000, 15,000, and 20,000 point locations. Table 5 summarizes the values used for each parameter in our experiments and their default values. We set the range for the area of the query rectangles as 0.001% to 0.01% of the total data space as this is a reasonable range of area to preserve a user's privacy (e.g., the range represents about 4 to 40 km^2 with respect to the total area of California).

Parameter	Range	Default
Group size	4, 16, 64, 256, 1024	64
Area of M	2%, 4%, 8%, 16%, 32%	8%
Query rectangle area	0.001% to 0.01%	0.005%
k	2, 4, 8, 16, 32	8
Synthetic data set size	5K, 10K, 15K, 20K	20K

Table 5: Experiment Setup

We consider 1000 private kGNN queries for each set of experiments, evaluate the proposed algorithms for each of these GNN queries and determine the average experimental results. We randomly generate 1000 point locations that are uniformly distributed in the total space. Each point p_q corresponds to a private kGNN query, where M is a rectangle centered at p_q. In each experiment the length and width of M are randomly generated for the given area of M.

For each private kGNN query, we randomly generate a point location within M for each user in the group. Then the query rectangle for each user is also randomly generated in such a way that each query rectangle resides in M and includes the user's point location. While generating query rectangles for a private kGNN query, we ensure that at least there is one query rectangle that touches each edge of M.

We run the experiments on a desktop with a Pentium 2.40 GHz CPU and 2 GByte RAM. We present our experimental results of the private filter algorithms and kGNN queries with respect to a set of query rectangles in Section 7.1 and Section 7.2, respectively.

7.1 Comparison of Private Filter Algorithms

We evaluate and compare the final pruning private filter (FPPF) and the incremental pruning private filter (IPPF) in terms of computational and communication costs. We add the time spent by each user in the group for the private filter technique and the total time represents the computational cost for a group to filter the answers of a private kGNN query. We compare the communication cost in terms of answer set size; then the total communication cost by adding the size of the answer set that a coordinator and each user in the group have to send. In our experiments, since we consider $n > 2$, we use the direct communication method, i.e., each user directly sends the modified answer set to another randomly selected user in the group.

We present the experimental results in Sections 7.1.1 to 7.1.4 and then analyze these results in Section 7.1.5.

7.1.1 Effect of group size

Figure 5(a) shows the time required by FPPF and IPPF for different group sizes. For SUM, the time required by IPPF is always higher than that of FPPF and the ratio of the required time between IPPF and FPPF decreases from 5.0 to 2.0 for the increase of group size from 4 to 16 and then remains constant at 2.0. For MAX, the time required by IPPF is significantly higher than that of FPPF for a small group size (e.g., 9.0 times higher for the group size of 4), but with an increase of the group size the time required by FPPF is higher than that of IPPF.

We observe in Figure 5(b) that the communication cost of IPPF is always lower than that of FPPF for both SUM and MAX. The communication cost of IPPF is on average 1.9 and 2.0 times lower than that of FPPF for SUM and MAX, respectively.

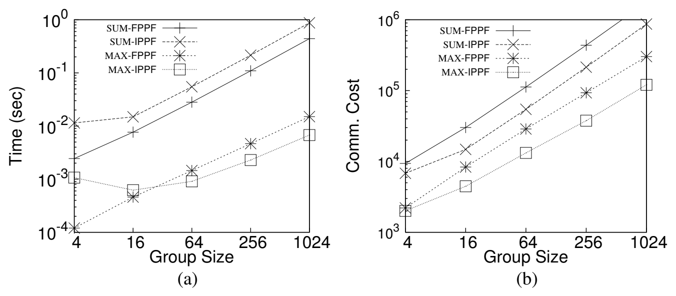


Figure 5: Effect of group size (data set *C*)

7.1.2 *Effect of the area of* M

We see in Figure 6(a) that the time required by IPPF is always 2.0 times higher than that of FPPF for every size of M in case of SUM. In case of MAX, the time for IPPF is nearly constant for any area of M, whereas the time required by FPPF first increases and then decreases with the increase of the area of M. We observe that IPPF requires more time than that of FPPF only for larger M.

Figure 6(b) shows that for SUM the communication cost of IPPF is approximately 2.0 times lower than that of FPPF for any area of M and for MAX the ratio of communication cost between FPPF and IPPF slightly decreases from 2.3 to 1.9 with the increase of the area of M.

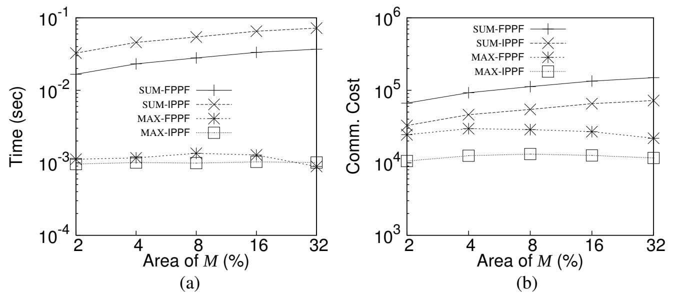


Figure 6: Effect of the area of *M* (data set *C*)

7.1.3 *Effect of query rectangle area*

Figure 7(a) shows that for SUM the time required by FPPF and IPPF for varying the query rectangle area follows a similar trend to that of varying the area of M, and for MAX the times of IPPF and FPPF both vary in a random manner with the increase of the query rectangle area and the time required by IPPF is never greater than that of FPPF.

Figure 7(b) shows that the communication cost of IPPF is on average 2.0 and 2.2 times lower than those of FPPF for SUM and MAX, respectively.

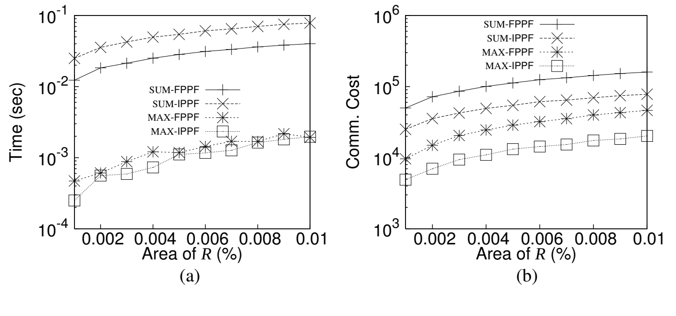


Figure 7: Effect of query rectangle area (data set *C*)

7.1.4 *Effect of* k

The effect of varying k is not significant for SUM as we see in Figure 8(a): the times for IPPF and FPPF remain nearly the same for different k and the required time of IPPF is on average 2.0 times higher than that of FPPF. For MAX the time required by FPPF is nearly constant and the time for IPPF slightly increases with the increase of query rectangle area and is equal to that of FPPF for $k = 32$.

We observe in Figure 8(b) that the ratio of the communication cost between FPPF and IPPF is approximately 2 for any k in case of SUM, whereas for MAX the ratio slightly decreases from 2.2 to 2.0 for increasing k from 2 to 32.

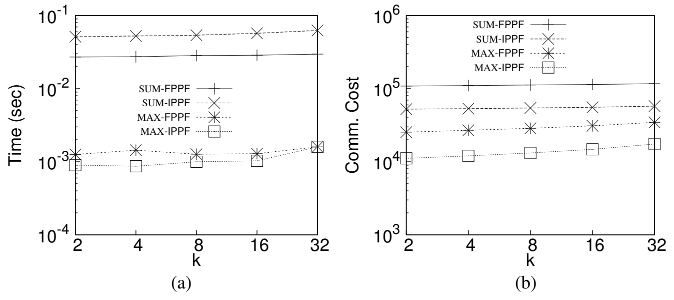


Figure 8: Effect of *k* (data set *C*)

7.1.5 *Comparative Analysis*

The experimental results for data sets U and Z also show a similar trend as data set C. In all experiments, the communication cost of IPPF is always lower (at least 1.9 times) than that of FPPF for both SUM and MAX. For the computational cost, we observe that in case of SUM, the computational cost of IPPF is always higher than that of FPPF, whereas for MAX the computational cost of IPPF is lower than that of FPPF in most of the cases.

The reason behind the higher communication cost of FPPF is that the answer set size remains constant in FPPF, whereas in IPPF the answer set size continuously reduces due to local pruning capability of each user. On the other hand, although in IPPF users process smaller answer sets and thereby reduce the computational cost, the local pruning adds extra computational overheads for each user. Moreover, the computational cost involved in local pruning is higher for SUM than that of MAX because in MAX, the users do not need to compute $maxdist_k$ (the k^{th} smallest maximum aggregate distance). From the experimental results we conclude that for SUM the required time for local pruning is higher than the reduction in time for processing smaller answer sets. For MAX the required time for local pruning is lower than the reduction of time for processing smaller answer sets in most of the cases and the opposite applies for the remaining cases.

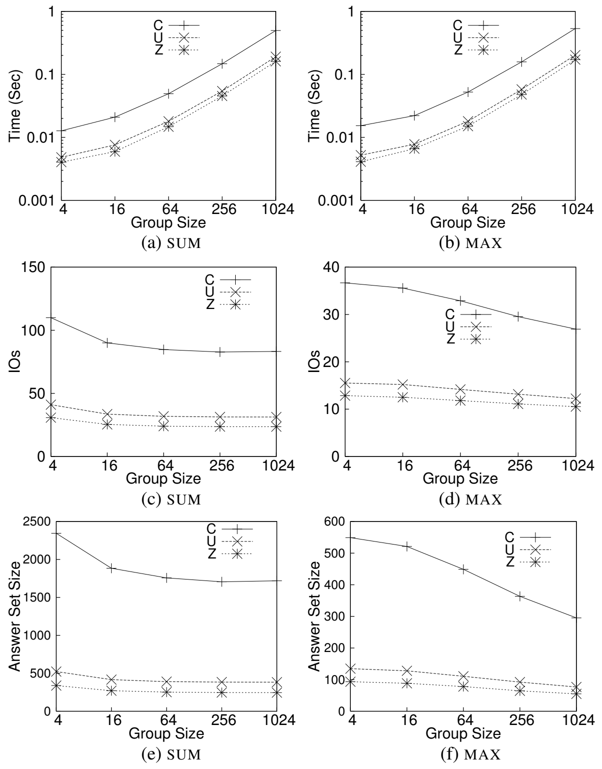
Note that we have designed our experiments independent of communication links used among the users, and shown the communication cost in terms of communication amount (i.e., answer set size). This allows us to approximate the communication delay from the known latency of the used communication link (e.g., wireless LANs, cellular link). Our proposed technique requires multiple rounds of communication, which may cause a delay in the response time. Nowadays this should not be a problem as the latency of wireless links has been significantly reduced, for example HSPA+ offers as low as 10ms latency. More importantly, a user might be happy to tolerate a reasonable delay to preserve her privacy.

7.2 Performance of *k*GNN Queries w.r.t. Rectangles

We evaluate the performance of our proposed algorithm REGION_kGNN in terms of the computational cost given by the processing time, the number of page accesses, i.e., IOs, and the candidate answer set size. In our experiments, the data points are indexed using an R^*-tree and the page size is set to 1 KB with a node capacity of 50 entries.

7.2.1 *Effect of group size*

We observe that the processing time increases with the increase of the group size (Figures 9(a) and (b)), because the larger the group size the larger the number of distance computations involved in computing an aggregated distance. On the other hand, Figures 9(c)-(f) show that both IOs and the answer set size decrease with an increase of the group size. The reason is as follows. We know that both minimum and maximum aggregate distances of a data point, i.e., d_{min} and d_{max}, increase or remain the same with the increase of the group size. For computing $maxdist_k$, only k data points or R^*-tree nodes with the minimum d_{max} are considered, whereas for d_{min} each data point or R^*-tree node is considered to test if it can be pruned. Hence, the probability is high that more d_{min} becomes larger than $maxdist_k$ with an increased group size.



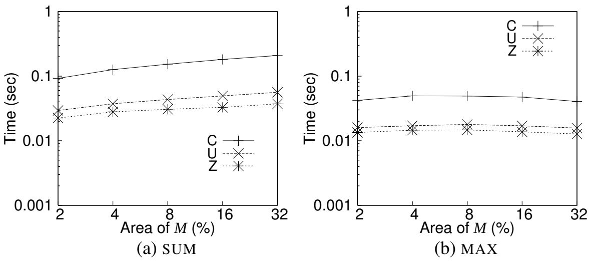
(a) SUM (b) MAX (c) SUM (d) MAX (e) SUM (f) MAX

Figure 9: Effect of group size

7.2.2 *Effect of the area of* M

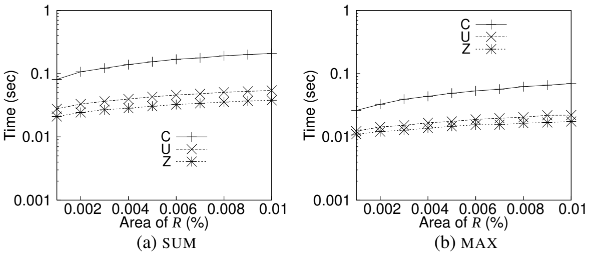
In this experiment we find that with an increasing area of M, the processing time, IOs, and the answer set size increase for SUM, and all of them first increase and then decrease for MAX. Due to space limitations, we only show the results for the required time in Figure 10.

There are two factors that influence the outcome of these experiments. Both d_{min} and d_{max} of data points or R^*-tree nodes that were outside a smaller M decrease or remain the same with a larger area of M, and thus these data points or R^*-tree nodes might not be pruned for a larger M. On the other hand, both d_{min} and d_{max} of data points or R^*-tree nodes that were inside of a smaller M decrease or remain the same with a larger M and hence these data points or R^*-tree nodes might be pruned for a larger M. In summary, if the former factor dominates, it results in an increase of the processing time, IOs, and the answer set size, and if the latter one dominates, it results in a decrease.



(a) SUM (b) MAX

Figure 10: Effect of the area of M



(a) SUM (b) MAX

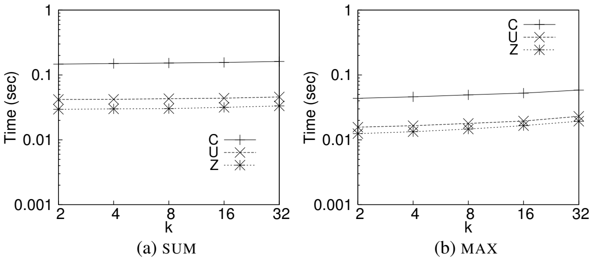
Figure 11: Effect of query rectangle area

7.2.3 *Effect of query rectangle area*

We find that the processing time, IOs, and the answer set size increase with larger query rectangles. With the increase of query rectangle area, for each data point, d_{min} decreases or remains the same whereas d_{max} does not decrease, i.e., less data points or R^*-tree nodes are pruned for larger query rectangle areas. Again, less pruning results in more distance computations and increases the processing time (Figure 11).

7.2.4 *Effect of* k

We expect that as $maxdist_k$ increases with the increase of k, less R^*-tree nodes or data points will be pruned for a larger value of k. Experimental results also show that the processing time (Figure 12), IOs, and the answer set size slightly increase with the increase of k.

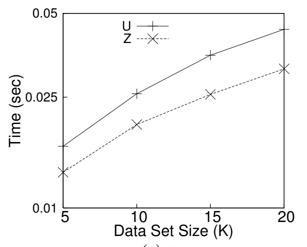


(a) SUM (b) MAX

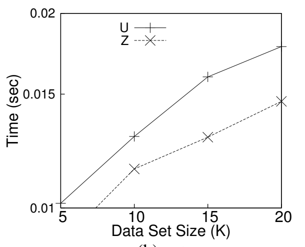
Figure 12: Effect of k

7.2.5 *Effect of data set size*

We observe that the processing time, IOs, and the answer set size increase for increasing data set sizes and the rate of increase



(a) SUM



(b) MAX

Figure 13: Effect of data set size

decreases for a larger data set. For example, the increase ratio of the processing time are 1.5 (SUM) and 1.1 (MAX) for increasing data set size from 5k to 10k whereas the increase ratio of the processing time are 1.2 (SUM) and 1.1 (MAX) for the increase of data set size from 15k to 20k (Figure 13).

For each set of experiments, except the experiments in Section 7.1.3 and 7.2.3, we also consider the case, where the users of a group have variable privacy levels, i.e., the area of query rectangles are different for a group. We find that the experimental results show similar trends to those for equally-sized query rectangles.

From experimental results, we conclude that our technique for private *k*GNN queries is scalable as it can cope with a very large group size (up to 1024) and we find that the processing cost slightly increases with the increase of user privacy level, i.e., the area of a query rectangle.

8. CONCLUSION

In this paper, we proposed a framework for privacy preserving group nearest neighbor queries. We addressed the problem of private *k*GNN queries in two parts: we developed private filter techniques that ensure privacy while computing the actual GNNs from a set of candidate answers for any group size (except group size 2 for MAX), and we proposed an algorithm for evaluating a *k*GNN query with respect to a set of regions. We considered two aggregate functions, SUM and MAX, that enable users of LBSs to meet at a point with the smallest total travel distance or to meet within the shortest time by minimizing the maximum distance, respectively.

Our experimental results show the performance analysis of our algorithm for different settings of privacy parameters. We compare two of our proposed private filter techniques: FPPF and IPPF. We find that FPPF incurs higher communication overhead than IPPF. On the other hand, in terms of computational cost, FPPF always performs better than IPPF for SUM, and IPPF performs better than FPPF for MAX in most of the cases. We also observe that our algorithm for *k*GNN queries with respect to a set of rectangles is highly scalable and ensures high privacy level with less processing overheads. To the best of our knowledge, this is the first work to address the problem of preserving user privacy for GNN queries.

In the future, we intend to investigate the possibility of reducing the number of candidate answers for *k*GNN queries with respect to a set of regions and aim to address the privacy issues for *k*GNN queries in road networks. We will also explore to what extent secure multi-party computations (e.g., [6]) can be used to enhance our approach.

9. ACKNOWLEDGEMENT

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

10. REFERENCES

[1] Location-based mobile social networking. http://www.abiresearch.com/research/1001722-Location-Based+Mobile+Social+Networking, 2008.

[2] Loopt. http://www.loopt.com, 2008.

[3] Privacy concerns a major roadblock for location-based services says survey. http://www.govtech.com/gt/articles/104064, 2007.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

[5] C. Bettini, S. Mascetti, X. S. Wang, and S. Jajodia. Anonymity in location-based services: Towards a general framework. In *MDM*, pages 69–76, 2007.

[6] D. Bickson, D. Dolev, G. Bezman, and B. Pinkas. Peer-to-peer secure multi-party numerical computation. *IEEE P2P*, 0:257–266, 2008.

[7] C.-Y. Chow, M. F. Mokbel, and X. Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In *ACMGIS*, pages 171–178, 2006.

[8] B. Gedik and L. Liu. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE TMC*, 7(1):1–18, 2008.

[9] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD*, pages 121–132, 2008.

[10] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Mobihide: A mobile peer-to-peer system for anonymous location-based queries. In *SSTD*, pages 221–238, 2007.

[11] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVÉ: Anonymous location-based queries in distributed mobile systems. In *WWW*, pages 371–389, 2007.

[12] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*, pages 31–42, 2003.

[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[14] T. Hashem and L. Kulik. Safeguarding location privacy in wireless ad-hoc networks. In *Ubicomp*, pages 372–390, 2007.

[15] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD*, pages 83–95, 1995.

[16] H. Hu and J. Xu. Non-exposure location anonymity. In *ICDE*, pages 1120–1131, 2009.

[17] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD*, pages 239–257, 2007.

[18] H. Li, H. Lu, B. Huang, and Z. Huang. Two ellipse-based pruning methods for group nearest neighbor queries. In *GIS*, pages 192–199, 2005.

[19] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *VLDB*, pages 763–774, 2006.

[20] R. Muntz, T. Barclay, J. Dozier, C. Faloutsos, A. Maceachren, J. Martin, C. Pancake, and M. Satyanarayanan. *IT Roadmap to a Geospatial Future*. The National Academies Press, 2003.

[21] S. Namnandorj, H. Chen, K. Furuse, and N. Ohbo. Efficient bounds in finding aggregate nearest neighbors. In *DEXA*, pages 693–700, 2008.

[22] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, page 301, 2004.

[23] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2):529–576, 2005.

[24] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[25] M. Strassman and C. Collier. Case study: The development of the find friends application. In *Location-Based Services*, pages 27–40. 2004.

[26] K. F. Yanmin Luo, Hanxiong Chen and N. Ohbo. Efficient methods in finding aggregate nearest neighbor by projection-based filtering. In *ICCSA*, pages 821–833, 2007.

[27] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE*, pages 366–375, 2008.