World Wide Web (2019) 22:375–416
https://doi.org/10.1007/s11280-018-0570-5

Protecting privacy for distance and rank based group nearest neighbor queries

Tanzima Hashem[1] · Lars Kulik[2] · Kotagiri Ramamohanarao[2] · Rui Zhang[2] · Subarna Chowdhury Soma[1]

Received: 27 March 2017 / Revised: 24 January 2018 / Accepted: 9 April 2018 /
Published online: 15 June 2018

Abstract This paper proposes a novel approach to safeguarding location privacy for GNN (group nearest neighbor) queries. Given the locations of a group of dispersed users, the GNN query returns the location that minimizes the total or the maximal distance for all group users. The returned location is typically a meeting place such as a cinema or coffee shop where the group would like to meet. In our work, we highlight the challenges for private GNN queries and propose a general framework that have two key features: (i) it ensures privacy in a decentralized manner and (ii) can compute an optimal location for GNN query that maximizes the group's overall preference for the meeting place. To mask their precise locations, we assume that user locations are given as regions to a location-based service provider (LSP). The LSP computes then a set of candidate answers (i.e., meeting

✉ Tanzima Hashem
tanzimahashem@cse.buet.ac.bd

Lars Kulik
lkulik@unimelb.edu.au

Kotagiri Ramamohanarao
kotagiri@unimelb.edu.au

Rui Zhang
rui.zhang@unimelb.edu.au

Subarna Chowdhury Soma
subarna089@gmail.com

[1] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh

[2] Department of Computing and Information System, University of Melbourne, Melbourne, VIC 3010, Australia

places) for the GNN query. We identify two privacy attacks on the user locations, the distance intersection attack and the rank disclosure attack. These attacks are possible when the answer of a GNN query is determined from the candidate answers in a straightforward manner. We develop private filters that prevent these attacks and compute the GNN from the retrieved candidate answers. Our decentralized approach ensures that neither the users nor the LSP can learn the location of any group member. Our algorithms compute from the candidate set an optimal meeting place given the group members' imprecise locations. Our key insight to an efficient computation is to prune the meeting places that cannot be GNNs given the locations of the group members within the search region. A comprehensive experimental evaluation shows the effectiveness of our approach to answering private GNN queries.

Keywords Group nearest neighbor queries · Location based services · Privacy

1 Introduction

The advent of social networking sites such as Facebook [9], Google+ [17], Loopt [39], Friend Finder [49] has enabled location-based services (LBSs) for a group instead of a single user. Consider a scenario, where a group of friends decide to have their dinner together while participating in a chat using Facebook. The group may want to meet at a restaurant that minimizes their total travel distance to the restaurant. Facebook allows users to share their locations with others. Thus, it is not hard to imagine that Facebook or a third party application in Facebook can offer an LBS to find the suitable nearest restaurant for the group based on user locations. A group nearest neighbor (GNN) query allows friends to meet at their nearest place such as restaurant, shopping center, pizza place or movie theater. In military applications, a GNN query can be used if a group of soldiers wants to meet in the shortest possible time.

Along with the benefits, location-based applications also bring privacy concerns among the users [7, 11]. For example, if a group of students wants to meet at the nearest library and one of them is located at a student counseling center at that time then this student might not want to reveal that location to others. The fact that a student is stressed or seeks professional help is personal information that the student might not want to disclose. Recently, Microsoft has conducted a survey [41] to assess the state of LBSs. The study showed that though 92% of the users consider LBSs as valuable services, 52% of the users are worried about privacy risks that could arise from accessing LBSs. The main concerns of the users come from sharing location data with unspecified organizations or people. Nowadays, not only large corporations like Google or Microsoft offer LBSs but also small developers, companies or employers can run their location-based applications via Apple's iPhone OS and Google's Android operating systems. The privacy threat is exacerbated if the available information from the access of LBS is linked with other data sources; in the example, if the location-based service provider (LSP) corroborates with the counseling center, then more specific information about the user could be revealed. Due to an increasing awareness of privacy risks, users might refrain from using LBSs, which would hinder the use of these valuable services [36, 51].

Existing research mainly focuses on developing techniques (e.g., [5, 20, 42, 48, 54]) that preserve user privacy during the request of nearest neighbor (NN) queries. These researchers address the problem of answering GNN queries while preserving privacy for all group members, which we call the *private GNN query*. Our approach for private GNN queries does

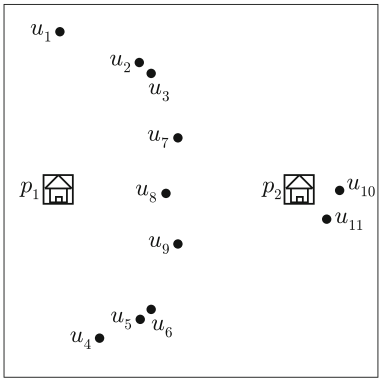
not disclose a user's location to others including the LSP and other group members thus ensuring privacy all the participants of LBS.

Formally, in a GNN query [44, 45], 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. Depending upon the application scenarios, the aggregate functions could be SUM and MAX, which return the total distance and the maximum distance from the users to the meeting location. Minimizing the total distance allows users to optimize their combined travel costs and minimizing the maximum distance enables users to meet at the shortest possible time.

In this paper, we introduce a new type of GNN query, a *GNN query based on a weighted-rank* (W-RANK), that maximizes the overall preference of the group. In a GNN query based on W-RANK, the meeting places are independently ranked by the users of the group and a set of weights are set by the group, where a weight is associated with each rank to express a group's level of cost for a meeting place with that rank. The meeting place that has the minimum total weighted-rank is considered as the GNN for the group. A weight representing the group's cost increases for a lower rank meeting place and is inverse to the group's preference. We formally define the GNN query based on W-RANK in Section 2.

The ranking of meeting places can be done using a single or multiple criteria, for example, the travel distance or food choice could be the factors affecting the ranking of the restaurants. A user's cost for traveling to a meeting place such as a restaurant is minimal if the other group members come at her nearest restaurant; the cost might be the time or the transportation cost to reach the meeting place. Thus, every user has the highest preference for the closest meeting place. A location is less preferred if the group decides to meet at the user's second nearest restaurant instead of the first one. In this paper, we focus on rankings that are based on the proximity of meeting places, an important factor for users for selecting any type of meeting place, since protecting user privacy for GNN queries based on proximity ranking is the most challenging among different criteria.

Figure 1a shows an example scenario, where $\{u_1, u_2, \ldots, u_{11}\}$ are user locations, and p_1 and p_2 are two meeting places. Figure 1b shows the distances of $u_1, u_2, \ldots, u_{11}$ to the meeting places. A GNN query that is based on traditional aggregate functions SUM or MAX and minimizes the total and maximum distance of the group to the meeting place, respectively, returns p_2 as the answer. On the other hand, p_1 is the closest to the majority



(a)

	p_1	p_2
u_1	4	7.6
u_2	4	5.5
u_3	4	5
u_4	4	6.5
u_5	4	5.5
u_6	4	5
u_7	3.4	3.6
u_8	2.7	3.7
u_9	3.4	3.6
u_{10}	8	1.5
u_{11}	7.5	1

(b)

Figure 1 An example scenario, where (**a**) $\{u_1, u_2, \ldots, u_{11}\}$ are user locations, and p_1 and p_2 are two meeting places, and (**b**) the distances of $u_1, u_2, \ldots, u_{11}$ to the meeting places

(i.e., 9 users u_1 to u_9) of the users. Thus, a GNN query based on W-RANK returns p_1 as the answer. A group may select p_1 over p_2 since p_1 maximizes the preferences of the group, and as a result the total and the maximum travel distance of the group increases slightly. The query answer varies depending on the users involved in a group. For example, if a group consists of users $\{u_1, u_2, \ldots, u_9\}$ then the returned answer is same for a GNN query based on SUM, MAX and W-RANK. There are also scenarios, where the answer of a GNN query based on W-RANK and SUM is the same but different from the answer based on MAX, e.g., for the group $\{u_2, u_2, \ldots, u_{10}\}$, and vice versa, the answer based on W-RANK and MAX is different to SUM, e.g., for the group $\{u_1, u_3, u_6, u_7, u_9, u_{11}\}$. In summary, depending on the user locations in a group, a GNN query based on W-RANK can return the same or different answer from a GNN query based on aggregate functions SUM and MAX but always maximizes the overall preference of the group. Thus, a group can request a GNN query based on W-RANK, SUM or MAX to find the appropriate meeting place according to their requirements. In this paper, we develop privacy preserving approaches for GNN queries based on SUM and MAX, and W-RANK.

1.1 Research problem

Recent research (e.g., [12, 18, 20]) has shown that sending a user's imprecise instead of exact location to the LSP while accessing LBSs is an effective model of protecting a user's location privacy from the LSP. Usually, the LSP returns a candidate answer set to the user that includes the answer for a LBS (e.g., a nearest neighbor (NN) query) for every location of the imprecise location. The user finds the query answer for her actual location from the candidate answer set. However, this straightforward technique does not work if the answer of a location-based query like a GNN query depends on the actual locations of a group of users instead of a single user. More specifically, if the LSP returns a candidate answer set to the group that includes the answer for a GNN query based on the group's required aggregate function (SUM, MAX or W-RANK) considering that a group member's location can be anywhere within her imprecise location, none of the group members can identify the actual GNNs without knowing the locations of other members.

Not only the LSP, but also 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 from a group of co-workers. To ensure a user's privacy, no involved party should have access to the locations of others. Thus the key research challenge in a private GNN query is to determine the actual GNN without allowing the LSP or other members to determine locations of users in the group.

1.2 Contribution

We identify two new privacy attacks: *distance intersection attack* and *rank disclosure attack* on a user's location while processing a private GNN query. In particular, we will show in Section 5.1 that a straightforward technique to determine the actual GNN based on SUM and MAX that does not share the actual user locations with other users, is prone to a privacy attack, which we call the distance intersection attack. In this technique, each user updates the distances 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. A user is located at the intersection

point of three circles having centers at the locations of any three candidate answers and radii equal to the user's distances from the locations of the corresponding candidate answers. We propose a *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.

For a GNN query based on W-RANK, the actual GNNs are determined from the candidate answer set based on group members' ranks instead of distances for the candidate answers. We will show in Section 7.1 that if group members straightforwardly reveal their ranks for the candidate answers, where the ranks are computed based on user distances to the meeting places, then the group members' locations can be approximated using the concept of Voronoi diagram [10]. We call such an attack on the user's location the rank disclosure attack. If the nearest meeting place of a user is known then the user's location can be refined in the area represented by the Voronoi cell of the meeting place within the Voronoi diagram. If the meeting places with the next ranks (e.g., the second nearest, the third nearest) are also known than the user's location can be further refined using a higher order Voronoi diagram.

Similar to GNN queries based on SUM and MAX, we propose a private filter technique for a GNN query based on W-RANK that prevents the rank disclosure attack by updating group members' ranks for the candidate answers in an aggregate form. From the aggregate form, no party can identify a user's ranks for the candidate answers and thereby approximate the user's location. To date, there exists no algorithm for an LSP to evaluate a GNN query based on W-RANK even for a set of point locations. Thus, an additional challenge for a GNN query based on W-RANK is to develop an algorithm for the LSP to evaluate a GNN query based on W-RANK with respect to a set of regions.

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

- Proposal of a framework to preserve each group member's privacy during the access of LBSs as a group. The unique advantage of our framework is its decentralized architecture, which does not require any trusted intermediary server.
- Provision of novel techniques that find the actual GNN based on aggregate functions SUM and MAX 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 [45] for point locations to regions, which is a necessary component to provide the candidate GNNs efficiently while preserving the privacy of group members.
- Introduction of GNN queries based on W-RANK and a privacy preserving solution to access the query. We develop a private filter technique that overcomes the rank disclosure attack while finding the GNNs from the candidate answer set. We propose an algorithm for the LSP to evaluate GNN queries based on W-RANK with respect to a set of regions.
- Evaluation of our techniques in extensive experiments and showing its efficacy.

This paper extends our previous paper [21], where we proposed the first privacy preserving approach to access GNN queries using the aggregate functions SUM and MAX. In this paper, we extend our work by introducing GNN queries based on W-RANK that maximize the level of preference for the group. We present an approach to protect user privacy while accessing GNN queries based on W-RANK. More specifically, (i) we identify the rank disclosure attack on a user locations that arise if the users disclose their ranks for meeting locations to others, (ii) we develop algorithms to prevent the attack and to process GNN

queries based on W-RANK with respect to a set of regions, and (iii) we present a new set of experimental analysis to verify the efficiency of our algorithms.

2 Problem setup

We formally define GNN queries in Section 2.1 and give an overview of privacy preserving GNN queries in Section 2.2. In Section 2.3, we discuss the privacy model including possible adversaries and privacy attacks for GNN queries.

2.1 GNN queries

We assume a system architecture where the users and the LSP are connected through a network (e.g., the cellular network or the Internet). 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) such as a cinema or coffee shop. The formal definition of the GNN query is given below:

Definition 1 (**GNN Query**) Let D be a set of data points in a 2-dimensional space, Q be a set of n query points $\{l_1, l_2, \ldots, l_n\}$ and f be an aggregate function. The GNN query finds a data point p from D, such that for any $p' \in D - \{p\}$, $f(Q, p) \leq f(Q, p')$.

A GNN query can be extended to a *k group nearest neighbor (kGNN) query* that returns k data points having the k smallest aggregate distances. For example, a group may want to know the locations of k group nearest restaurants and then select one from k options based on other parameters like budget or food preference. We develop privacy preserving solutions for kGNN queries.

In this paper, we first 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. Then we consider a GNN query based on weighted-rank (W-RANK), which is described below.

The aggregate function W-RANK returns the total weight of a group for a data point, where the total weight represents the cost associated with a group for a data point. The total weight of a data point is computed by adding the weights for all group members based on their ranks for the data point, where the ranks are computed based on user distances to the data points. A weight w_i is added to the total weight of a data point p_h if a user's rank for p_h is i. Since a user has the highest preference for her nearest data point, and a weight represents the cost associated with a rank of the data point, the weight increases with the decrease of the rank, i.e., $w_{i+1} > w_i$ (e.g., a user's nearest data point has the smallest rank 1 and the lowest cost w_1).

The group sets the rules to compute the set of weights according to their requirements. For example, the ratio between two consecutive weights w_{i+1} and w_i can be predefined; if it is set to 2 then $w_{i+1} = 2 \times w_i$. The ratio can be also predefined with respect to the weight w_1; if the ratio between w_1 and w_i is set to i, then w_i is computed as $i \times w_1$. The GNN query based on W-RANK returns the data point that minimizes the total weight (i.e., maximizes the preferences) for the group. For simplicity, in this paper, we assume that all group members use a same rule to compute the weights. In general, in a GNN query based on W-RANK, users can predefine their rules to compute weights independently according to their preferences.

Although distance is the most widely used metric to represent travel cost, in practice there are also cases where the cost is not *directly proportional* to the distance. For example,

many big cities (e.g., Melbourne) are divided into zones for the public transport system; if a user travels within a zone using a public transport then the transport cost is same irrespective of the actually traveled distance. Similarly, in a postal service, usually the local postage cost is same for any distance. There are scenarios where the cost for a train ticket increases with increasing number of crossed stations to reach the destination though the distances between two consecutive stations may not be the same. In this paper, we focus on weighted user ranks for the data points in addition to distance based aggregate functions SUM and MAX to provide the group with an option of maximizing their preferences for the data points.

Note that if the weight given by a user for a data point is equal to the distance of the user to the data point then W-RANK maps to SUM. Thus, the aggregate function W-RANK is a more generalized function; the weights can be used to represent costs for different metrics such as distance, rank, or the number of train stations a user needs to cross to reach at a data point.

2.2 Privacy preserving GNN queries

When a user requests a location-based query the user reveals her location, identity and the requested query to the LSP. From the revealed information the LSP can derive sensitive and private information about the user. A popular privacy model to hide a user's location from the LSP is to send a cloaked area [16, 52], which is typically a rectangle containing the user's location, instead of the user's location. The higher the spatial imprecision is, the more private is the user's location; the probability is higher that a larger rectangle area contains a diverse range of locations. On the other hand, if a user wants to hide her identity then the rectangle is computed based on K-anonymity [12, 18], which ensures that the rectangle includes the locations of at least $K - 1$ other users in addition to the user's location so that the user becomes K-anonymous.

In our proposed privacy preserving (private) kGNN queries, the users in the group do not reveal their exact locations and identities to the LSP; instead they provide rectangles $R_1, R_2, \ldots, R_n$ that contain user actual locations $l_1, l_2, \ldots, l_n$, respectively in addition to the $K-1$ other user locations. Thus the LSP needs to return a set of candidate data points A with respect to the provided rectangles that include the k GNNs for the actual user locations. Afterwards the actual k GNNs has to be computed from A without revealing the location of any user to any group member.

As we have seen in Section 1, providing an answer for privacy preserving kGNN queries has two parts: (i) The group of users use a technique, called a *private filter* (privacy preserving 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 rectangles to provide the candidate answers for the private filter while preserving user privacy. We formally define the private filter and the kGNN query with respect to rectangles as follows.

Definition 2 (**Private Filter**) Let $\{u_1, u_2, \ldots, u_n\}$ be a group of n users, A be a set of candidate data points, 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 and provides k smallest values for f with respect to the set $\{l_1, l_2, \ldots, l_n\}$ without revealing a user u_i's location l_i to others.

Definition 3 (k**GNN Query with respect to Rectangles**) 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 rectangles, r_i be any point in R_i for $1 \leq i \leq n$, and f be an aggregate function. The kGNN query with respect to rectangles

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 $\{r_1, r_2, \ldots, r_n\}$.

The symbols used in this paper are summarized in Table 1.

2.3 Privacy model: adversaries and privacy attacks

We consider the LSP and the users involved in processing kGNN queries as potential adversaries. In our approach, the group reveals users' imprecise locations (i.e., rectangles) to the LSP anonymously. On the other hand, since kGNN queries are used by the users to meet at a place, we assume that a user is not concerned to hide her identity from other group members. Instead the user does not disclose her location in any form to other group members to protect her privacy. We make the following assumptions for the privacy model in our approach:

- The LSP and the group members do not collude with each other to violate a user's privacy. We assume a semi-honest model, where an involved entity in the kGNN query processing follows the rules of our proposed approach and tries to infer private information of other members only from the obtained information during the execution of our proposed approach.
- The adversaries do not have any background information about user locations and do not know any probabilistic distribution of users locations.

A privacy attack, in general, refers to an intrusive inference through which an adversary is able to derive a user's sensitive and private information that the user does not want to reveal. For example, if a user reveals a rectangle as her location and an adversary can refine the user's location within the rectangle then an inference attack can be successful. The privacy attacks that the LSP can apply to violate user privacy are already identified and addressed in the literature. We use the approach proposed in [20] to hide user identities and locations from the LSP. In [20], each user computes a rectangle that includes $K - 1$ other users' rectangles in addition to the user's exact location, and anonymously sends to the LSP. In Section 3.1, we elaborate the superiority of [20] over other existing approaches for hiding user identities and locations from the LSP. In addition, we propose efficient algorithms for the LSP to evaluate the candidate k GNNs with respect to a set of rectangles.

In this paper, we focus on how to protect a user's location privacy from others while finding the actual k GNNs from the returned candidate answer set by the LSP. In Sections 5.1 and 7.1, we show that if a user straightforwardly reveals her distance or rank for the candidate data points then other group members can apply the distance intersection attack or the rank disclosure attack, respectively, to identify the user's location. We have developed

Table 1 Notations

Symbol	Meaning
$\{u_1, u_2, \ldots, u_n\}$	A group of n users
l_i	The location of a user u_i
R_i	The rectangle of a user u_i
k	The number of required GNNs k
p_h	The location of a data point
A	A set of candidate data points that include k GNNs
f	An aggregate function SUM, MAX or W-RANK

private filter techniques that prevent these attacks and protect user privacy while processing kGNN queries.

3 Related work

In Section 3.1, we present the solutions for protecting location privacy of users accessing LBSs, and in Section 3.2, we discuss the existing work related to GNN queries.

3.1 User privacy in LBSs

Most research (e.g., [12, 18, 42, 53]) to preserve user privacy in LBSs is based on a centralized architecture, where an intermediary trusted server performs all tasks for the privacy protection of users. The limitations of a centralized architecture is a single point of failure, bottlenecks due to communication overheads, and privacy threats as the intermediary server stores all information in a single place. Therefore, a few decentralized approaches (e.g., [8, 13, 20, 22]) preserve a user's privacy in cooperation with her peers and do not involve an intermediary trusted server. However, all approaches, centralized or decentralized, focused on LBSs involving a single user.

A centralized architecture for LBSs involving a group like private GNN queries can be implemented in a similar way, where an intermediary trusted server transforms exact locations of users to regions (e.g., a rectangle or a circle) and forwards the GNN query with regions instead of the exact user locations to the LSP. The LSP evaluates the candidate answers that include GNNs for any position of the users within their regions and returns the candidate answers to the intermediary server. The intermediary server computes the actual GNN for the exact locations of the users from the candidate answers and forwards the actual answer to the group. To overcome the limitations of a centralized architecture, we propose a framework based on a decentralized architecture for a private GNN query.

Most of the decentralized techniques (e.g., [8, 13, 14, 20, 29]) hide a user's location from the LSP by exploiting P2P network (e.g., Bluetooth, WiFi). In these techniques, an imprecise location of the user (i.e., a rectangle or a circle) 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 [8, 13, 14], the users need to trust their peers with their locations for computing their imprecise locations.

In [29], K users in the proximity form a group and then using the proximity information, the group progressively finds a bounding box as their rectangle that covers all users' locations. In this technique, all users in a group use the same rectangle for accessing LBSs. Though the users do not share their actual locations with anyone, not even their peers, a user's rectangle is revealed to others in the group. Since our proposed private filter requires that the user's rectangle sent to the LSP is not revealed to anyone, and therefore we do not use the technique proposed in [29] to compute the user's rectangle for a private kGNN query.

The solution proposed in [20] is a two step process. In the first step, a local imprecise location (i.e., a rectangle) is randomly computed by each user such that the rectangle includes the user's location and the area of the rectangle is set according to the user's privacy requirement. In the second step, when a user requires to access a LBS, the user's global imprecise location for the LSP is computed as a minimum bounding rectangle that includes $K-1$ others' local imprecise locations (i.e., rectangles) in addition to the user's exact location. Since neither the user's actual location nor the rectangle sent to the LSP is revealed to others, we use this technique to request a private GNN query.

Besides K-anonymity [50] and imprecision [25], space transformation [33] and private information retrieval techniques [15] are also used to preserve the privacy of the users. Although the architecture of both of these approaches eliminates an intermediary trusted server, these approaches incur cryptographic overheads and cannot use existing spatial indexing techniques to store data on the server. In this paper, we assume that the users in a group disclose their imprecise locations to the LSP, which evaluates the query based on these imprecise locations on a database, where the data points are indexed using a spatial indexing technique such as R-tree [19].

In [23], the authors have proposed a privacy preserving approach that eliminates the role of the LSP, and evaluates GNN queries with crowd sourced data points. However, the limitation of this approach is that the considered POI set is small and can be incomplete. Thus, the group nearest data points identified with this approach may not be the actual GNNs. Recently three cryptographic approaches, [2, 3, 30], have been developed to address user privacy for GNN queries. In [2, 3], the authors have developed a cryptographic technique to find the centroid of all group members' locations and the LSP evaluates the GNN query with respect to the centroid. Thus, the LSP cannot return the actual group nearest data point with respect to the group, instead the LSP has to approximate the query answer, which is the main limitation of their approach. On the other hand, in [30], the authors have not provided any algorithm to evaluate the candidate answer set for a GNN query from which the group can find the actual query answer. Cryptographic approaches, in general, incur high processing overheads. We develop a lightweight approach to evaluate GNN queries in a privacy preserving manner. 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

In [44], Papadias et al. first proposed kGNN queries for the aggregate function SUM. In [45], the authors have extended the proposed solution [44] for GNN queries for minimizing the minimum and maximum distance of group members in addition to minimizing the total distance of group members, and called the GNN query as the aggregate nearest neighbor query. Both depth first search (DFS) [47] and best first search (BFS) [27] algorithms can be used to implement the proposed solutions [44, 45], where R-tree [19] is used to index the data points.

Among the three methods, MQM (multiple query method), SPM (single point method) and MBM (minimum bounding method), shown in [44, 45], MBM outperforms both MQM and SPM in terms of the query processing overhead. The key idea of MBM is to traverse the R-tree in order of the minimum aggregate distances of R-tree nodes with respect to the set of query points. The search terminates when k data points with k smallest aggregate distances have been identified. To increase the efficiency of MBM, the authors have proposed strategies to prune the R-tree nodes/data points using the minimum bounding box of the set of query points.

In [34], 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 [40], Luo et al. have only considered non-indexed data points and developed a GNN algorithm based on projection-based pruning strategies, and in [43], Namnandorj et al. have estimated the search space using a vector property and proposed GNN algorithms for both indexed and non-indexed data points. In [35], Li et al. have developed exact and approximation algorithms for GNN queries that minimize the

maximum distance of the data points from a set of points. The authors have used convex hull, minimum enclosing ball and furthest Voronoi diagram in their algorithms to prune R-tree nodes/data points. However, their work does not consider an important and popular aggregate function SUM that minimizes the total travel distance of the group. We do not extend the point-based GNN query evaluation algorithm [35] for the set of rectangles because it is not straightforward to extend the concept of furthest Voronoi diagram for rectangles.

Our 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. In our algorithm for GNN queries with respect to a set of rectangles for aggregate functions SUM and MAX, we extend the concept of the minimum bounding box proposed in [34, 44, 45] to prune R-tree nodes/data points. To the best of our knowledge, there exists no algorithm in the literature for processing GNN queries based on W-RANK with respect to a set of points or rectangles.

The work proposed in [37] considers a new type of GNN query that minimizes the total travel distance and the maximum distance for a fixed size of a subgroup. For example, a group may request for a data point that have the minimum total distance from any 60% users in the group. Formally, a group provides their locations and the subgroup size and the LSP returns the subgroup and the data point that result in an optimal answer for aggregate functions SUM and MAX. In this paper, we focus on GNN queries that involve all group members to determine the answer. In [38], the authors have proposed an efficient solution to evaluate GNN queries for moving groups, whereas in our work, we assume that the groups are static.

Besides GNN queries, researchers have also focused on other applications that involve data from more than one user or entity like group trip planning queries [24, 26], group trip scheduling queries [32], flexible group spatial keyword queries [1], personalized app recommendation [46], distributed outlier detection [55], and emerging topic tracking in microblog stream [31].

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.

In our proposed framework a coordinator for the group is selected randomly before a query request. The coordinator assists in processing the private GNN 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 their 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 member sends her imprecise location and the QID anonymously to the LSP using either a pseudonym service [14] in the Internet or through a randomly selected peer [8, 20] 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 rectangles. 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.

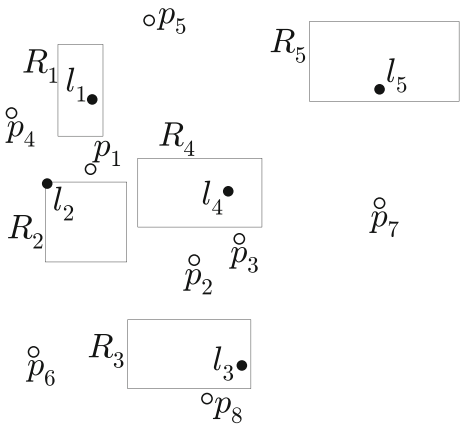
5 Private filters

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 that minimize the total travel distance with respect to the actual locations $\{l_1, l_2, \ldots, l_5\}$ of the users (Please see Section 6 for the algorithm to compute A). The locations of the data points in A, the actual and imprecise locations of the users are shown in Figure 2, and the actual distances of the users to all data points in A are presented in Table 2.

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

Figure 2 An example scenario



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 3). 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.

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 intersection points 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.

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 3). In the case that the communication among the group members is done via a coordinator, 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

Table 2 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	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 3 Updated distances after adding each user's actual distance 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

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 3.

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:

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

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:

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

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

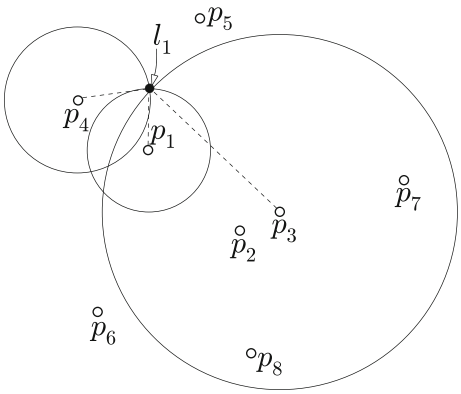


Figure 3 An example of distance intersection attack

and updated A. In every step of the private filter technique, $d_{max}(p_h)$ can be in general expressed by the following equation:

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

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 4 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 4). 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.

In this technique, the privacy of all users is preserved in both scenarios: communication 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 the 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 (3). In the second scenario, although c can monitor the change in d_{max}s for the 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 4). However, as the location of R_4 is unknown to c, c cannot determine $MaxDist(R_4, p_4)$ to compute $Dist(l_4, p_4)$. To remind the readers, as we already mentioned in Section 2.3, we assume that the LSP and the group members do not corroborate among themselves to retrieve any extra information about a user; our approach is based on a semi-honest model, where an involved entity in the kGNN query processing follows the rules of our proposed approach and tries to infer private information of other members only from the obtained information during the execution of our proposed approach.

The private filter technique discussed so far cannot perform any pruning of the 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 the query rectangles d_{min} in addition to the sum of the maximum distances from the 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.

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

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

	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

Let $MinDist(R_i, p_h)$ be a function that returns the minimum Euclidean distance between R_i and p_h. The LSP computes $d_{min}(p_h)$ as follows:

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

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:

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

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

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 5 shows the steps for updating d_{min} and d_{max}, and the pruning of data points by every user in our example. From Table 5, 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.

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 GNNs 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 (5);
1.3 $d_{min}(p_h) \leftarrow d'_{min}(p_h)$;
1.4 compute $d'_{max}(p_h)$ using equation (2);
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;

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 (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 proof of correctness of our private filter is elaborated in [21].

5.2 Minimizing the maximum distance

In this section, we consider private *k*GNN 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:

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

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

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 $d_{min}(p_h)$.

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 the 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 the 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 4 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 4a, the intersection point of all circles does not refer to any user's location, and in Figure 4b 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 the 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

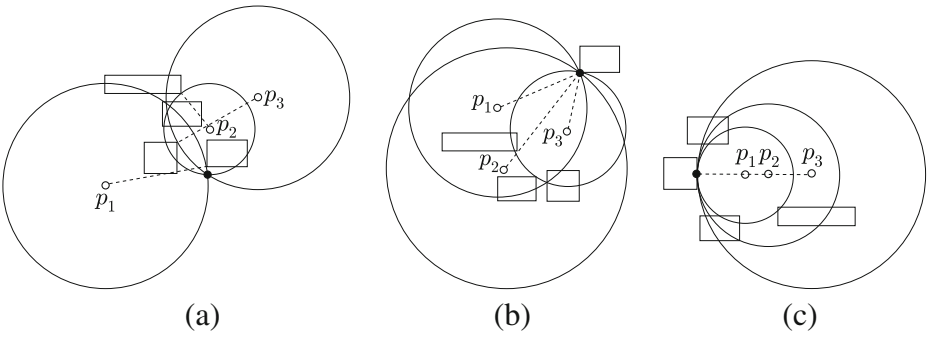


Figure 4 Examples scenarios of circles with radii equal to d_{min}s retrieved from the LSP

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 *k*GNN queries with respect to 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 $\{R_1, R_2, \ldots, R_n\}$.
- $MinDist(q, p)$ ($MaxDist(q, p)$): 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 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 as 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.

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. The following lemma describes the termination condition of the algorithm as no other data point can further qualify as a candidate answer once the condition is true.

Lemma 1 *Let p be a data point or an R^*-tree node dequeued from Q_p. The algorithm terminates if $d_{min}(p) > maxdist[k]$.*

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 [44, 45], the algorithm checks if p can be pruned according to the following lemma.

Lemma 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 Lemma 2, then the algorithm uses the tighter condition $d_{min}(p) > maxdist[k]$ of Lemma 1 to check if p can be discarded before inserting it into Q_p. 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 Lemma 2 but satisfies the condition of Lemma 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.

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

The detail pseudocode and the proof of correctness of the algorithm is shown in [21].

7 Private kGNN queries based on weighted-rank

In a private kGNN queries based on weighted-rank, a group of users provide their query rectangles and specify the number of required group nearest data points k, and a rule $\mathcal{R}$ to compute weights representing the costs associated with the ranks of the data points to the LSP. Let a weight w_i represent the cost associated with rank i for $i > 0$ and $w_{i+1} > w_i$. The weight w_i can be defined according to the preference of the group; for example, the rule $\mathcal{R}$ to compute the weights can be with respect to w_1 as $i \times w_1$ or with respect to w_{i-1} as $rt \times w_{i-1}$ for any positive integer $rt > 0$. The LSP evaluates the query and returns the set of data points A as candidate answers. To determine the actual GNN from the returned set

of data points by the LSP, each user in the group needs to reveal her ranks for the data points in the candidate answers. Since a user does not need to update her actual distance for each data point in the candidate answers for private kGNN queries based on W-RANK, there is no possibility of violating the user's privacy with the distance intersection attack. However, we will show that the user's privacy could be still violated with the *rank disclosure attack*, if the user straightforwardly provides her rank information.

In Section 7.1, we discuss the rank disclosure attack and in Section 7.2, we present a final pruning private filter technique to overcome the rank disclosure attack. In Section 7.3, we propose an algorithm to evaluate kGNN queries based on W-RANK with respect to a set of rectangles. In Section 7.4, we show why it is not possible to design an incremental pruning private filter technique for kGNN queries based on W-RANK.

7.1 Rank discloser attacks

Users need to disclose their ranks for data points in A to determine the actual k GNNs. Let the first user u_1 that receive A from the coordinator c, insert and initialize a weight with 0, denoted as $w(p_h)$, for each data point $p_h \in A$. Then the user finds her ranks for the data points in A and updates $w(p_h)$ according to the following equation.

$$w(p_h) = w_{rank(l_1, p_h)} \tag{8}$$

Note that, l_1 is the first user u_1's actual location and $rank(l_1, p_h)$ is the u_1's rank for p_h computed with respect to l_1; for example if p_h is the user's 3^{rd} nearest data point then $rank(l_1, p_h)$ is 3 and the cost associated with $rank(l_1, p_h)$, w_3, is assigned to $w(p_h)$. The answer set A is forwarded to the next user randomly either directly or via c. The process continues until all users have updated the w field for all data points in A. On receiving A, each user u_i computes $w'(p_h)$ for the data points with respect to her actual location l_i as follows:

$$w'(p_h) = w(p_h) + w_{rank(l_i, p_h)} \tag{9}$$

Then the user updates $w(p_h)$ for those data points by assigning $w'(p_h)$ to $w(p_h)$. After all users's updates, the value of $w(p_h)$ of a data point p_h represents the cost of the group for that data point, and the data point with the minimum weight (cost) is selected as the GNN. Similarly, the k data points in A with k smallest weights are selected as k GNNs.

Thus, if the users communicate directly, then the second user can know the first user's ranks for the data points by observing visited field of user identities. On the other hand, when the users communicate via c then c can monitor the change of $w(p_h)$ before sending and after receiving back from a user and thereby know each user's ranks for the data points. Learning the ranks for the data points of a user in the group enables the coordinator c or a group member to approximate the user's location by applying the *rank disclosure attack*.

The underlying idea of the rank disclosure attack on a user's location is based on the well known concept of ordered Voronoi diagram [10]. The first order Voronoi diagram for the data points is the division of the total space into subspaces, called Voronoi cells, where each Voronoi cell corresponds to a data point p_h such that p_h is the nearest data point for every location on that cell. Figure 5a shows an example of 1-order Voronoi Diagram of candidate answers $A : \{p_1, p_2, \ldots, p_8\}$. Since the location of candidate answers are known to the group member and c, if they also know the nearest data point of a user then they can easily identify the user's location as the Voronoi cell corresponding to that data point. For example, if p_2 is a user's nearest data point then the user must be located in the Voronoi cell shown with ash color in Figure 5a.

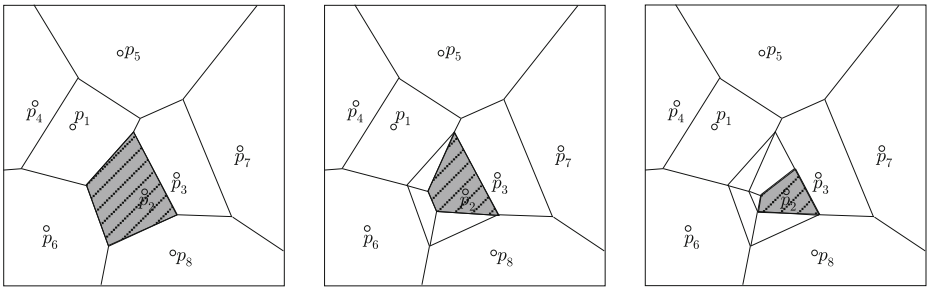


Figure 5 The data points $\{p_1, p_2, ..., p_8\}$ with (**a**) 1-order Voronoi diagram, (**b**) the 2-order Voronoi diagram, (**c**) 3-order Voronoi diagram

For an m-order Voronoi diagram, each Voronoi cell corresponds to a fixed rank for m data points. Figure 5b and c show 2-order Voronoi diagram and 3-order Voronoi diagram, respectively, for the same candidate answers used in Figure 5a. If p_2 and p_3 are a user's 1^{st} and 2^{nd} nearest data points, respectively, then the user must be located in the ash Voronoi cell shown in Figure 5b. Similarly, if p_2, p_3, and p_7 are a user's 1^{st}, 2^{nd}, and 3^{rd} nearest data points, respectively, then the user must be located in the ash Voronoi cell shown in Figure 5c. From the construction of order Voronoi diagram, we know that the Voronoi cells of higher order diagram are originated from the recursive division of Voronoi cells of lower order diagram. Thus, the more information about a user's rank is known (i.e., the larger the value for m), the more precise location of the user could be determined by constructing a higher order Voronoi diagram.

Algorithm 3 W-RANK_FPPF(R_i, l_i, k, rt, A)

Input : The user's rectangle R_i and exact point location l_i, the number of required GNNs k, and the answer set $A := \cup_h \{p_h, w(p_h)\}$

Output: Updated answer set A.

3.1 **for** *each* $\{p_h, w(p_h)\} \in A$ **do**

3.2 compute $w'(p_h)$ using equation (11);

3.3 $w(p_h) \leftarrow w'(p_h)$;

7.2 Private filter

To avoid the rank disclosure attack, similar to the kGNN queries for the aggregate function SUM, we propose a final pruning private filter (FPPF) technique for private kGNN queries based on W-RANK. We call the private filter as W-RANK_FPPF. For kGNN queries based on W-RANK, it is not possible to apply an incremental pruning private filter (IPPF) technique as IPPF cannot overcome the rank disclosure attack, which is discussed in Section 7.4. In W-RANK_FPPF, each user's update of rank information remains hidden in an aggregate form from others but at the end $w(p_h)$ represents the actual cost of p_h for the group.

For W-RANK_FPPF, in addition to returning the location of the set of data points as candidate answers, the LSP returns $w(p_h)$ for each data point $p_h \in A$. The LSP initializes $w(p_h)$ as the sum of the maximum distances of p_h to the query rectangles:

$$w(p_h) = \sum_{i=1}^{n} MaxDist(R_i, p_h) \tag{10}$$

Note that, R_i is a user u_i's query rectangle and n is the group size. On receiving A, each user computes $w'(p_h)$ as follows:

$$w'(p_h) = w(p_h) - MaxDist(R_i, p_h) + w_{rank(l_i, p_h)} \tag{11}$$

Then the user updates $w(p_h)$ for those data points by assigning $w'(p_h)$ to $w(p_h)$. Algorithm 3 summarizes the steps for W-RANK_FPPF.

Table 6 shows the ranks of the users for the data points in A shown in Figure 2. Table 7 shows the steps for updating $w(p_h)$ by every user for the given example, where without loss of generality, the rule $w_i = i \times w_1$ is used for $w_1 = 1$. After the updates of u_5, $w(p_1), w(p_2), \ldots, w(p_8)$ represent the actual aggregate weight (i.e., cost) of $\{p_1, p_2, \ldots, p_8\}$ of the group of users $\{u_1, u_2, \ldots, u_5\}$ (see the last row of Table 7). Depending on the communication method used, u_5 or the coordinator c forwards 2 GNNs, p_2 and p_1 (or p_3), to all users in the group.

In this private filter technique, similar to FPPF for the aggregate function SUM, an individual user's costs for the ranks of the data points remain hidden in aggregate form in both scenarios: communication without or with the coordinator. Since a group member or the coordinator cannot identify a user's added weights (i.e., cost) for the data points, the group member or the coordinator cannot also infer the ranks of the user for the data points. Thus, W-RANK_FPPF finds the GNNs without revealing the ranks of the users for the data points to others and prevents the rank discloser attack.

Let X represent a subset of users in the group who have already updated w 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, $w(p_h)$ can be in general expressed by the following equation:

$$w(p_h) = \sum_{u_x \in X} w_{rank(l_x, p_h)} + \sum_{u_y \in Y} MaxDist(R_y, p_h) \tag{12}$$

For a group size of $n = 2$, the user u_y who receives directly from the user u_x can know the identity of u_x by observing the list of entities and determine $w_{rank(l_x, p_h)}$ (thus, $rank(l_x, p_h)$), since u_y knows her own $MaxDist(R_y, p_h)$ in (12). Therefore, similar to the private filter for SUM, for $n = 2$, the users communicate via the coordinator to avoid the rank disclosure attack.

We omit the proof of correctness for W-RANK_FPPF due to its similarity with SUM_FPPF.

Note that for W-RANK_FPPF, the LSP does not provide the minimum and maximum weights with respect to the set of rectangles for each data point in the candidate answer set. As a result, users cannot perform any local pruning of the data points from the candidate answer set after updating the weight of each data point, which is a property of incremental

Table 6 Actual ranks of the users for the data points in A

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
u_1	1	4	5	2	3	6	7	8
u_2	1	3	5	2	5	3	8	7
u_3	6	2	3	7	8	4	4	1
u_4	3	2	1	7	5	8	4	6
u_5	5	4	2	6	3	8	1	6

Table 7 Updated $w(p_h)$ with respect to each user's actual rank for the data points in A

	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	42	39	37	58.5	53	60.5	53.5	54
u_2	39	34.5	33.5	52	46.5	55	47	49
u_3	33	30	30	43	38.5	49.5	37.5	46.5
u_4	28.5	26.5	27.5	38	34.5	44.5	31	42
u_5	16	15	16	24	24	29	24	28

private pruning filter (IPPF) technique. In Section 7.4, we will show the possible privacy attacks on a user's location that can arise from the IPPF technique for kGNN queries based on weighted-rank.

7.3 kGNN queries based on weighted-rank with respect to regions

In this section, we present an algorithm, REGION_kGNN_rank, to evaluate kGNN queries based on W-RANK with respect to a set of rectangles. The algorithm performs the evaluation in two parts. In the first part, the algorithm finds a set of data points that includes k GNNs based on W-RANK with respect to a set of rectangles. Then, in the second part, the algorithm computes the minimum and maximum weight of those data points with respect to the set of rectangles and prunes the data points based on the computed weights that cannot be the part of candidate answers. Algorithm 4, REGION_kGNN_rank, shows the evaluation process of kGNN queries based on W-RANK with respect to a set of rectangles. The input to Algorithm 4 are a set of rectangles $\{R_1, R_2, \ldots, R_n\}$, the number of required data points k, and the weights associated with ranks $\{w_1, w_2, \ldots\}$. The output is the candidate answers A with their weights, where a weight of a data point is the sum of the maximum distances of the data point to the rectangles.

In contrast to the algorithm for evaluating kGNN queries with respect to a set of rectangles for the aggregate function SUM and MAX in Section 6, the first part of Algorithm 4 cannot compute the candidate answer set directly, instead it computes a superset of the candidate answer set. In the algorithm for evaluating kGNN queries with respect to a set of rectangles for the aggregate function SUM and MAX in Section 6, at the same time of searching the data points on the database, the aggregate function SUM or MAX is used to prune the data points that cannot be candidate answers. In Algorithm 4, however, the weight of a data point cannot be computed before knowing the ranks of the data points for every rectangle. Thus, pruning the data points based on the weight is performed in the second part of Algorithm 4 (Line 4.37) using the function $Refine\_Answers$.

In the first part, the algorithm finds a set of data points that includes k nearest data points with respect to every point of each rectangle since a user can be located anywhere within her rectangle. Then the algorithm extends the search until a certain distance, i.e., $enddist$ (defined later in this section), that ensures that the set of data points include k GNNs based on W-RANK with respect to the set of rectangles. To evaluate k nearest data points with respect to every point of a rectangle, any existing algorithm [28, 42] can be used. However, executing those algorithms separately for each rectangle requires multiple searches on the database and incurs high IOs and query processing overhead. Our algorithm finds k nearest data points for all rectangles with a single search on the database. The first

part of Algorithm 4 (Lines 4.1-4.36) is based on a concept similar to the algorithm that evaluates GNN queries with respect to a set of rectangles for the aggregate function SUM and MAX.

Algorithm 4 REGION_kGNN_rank($\{R_1, R_2, \ldots, R_n\}, k, \mathcal{R}$)

Input : A set of rectangles $\{R_1, R_2, \ldots, R_n\}$, the number of required GNNs k, and a rule $\mathcal{R}$ to compute weights.
Output: A, a set of data points with their weights.

4.1 $A \leftarrow \emptyset$;
4.2 $end \leftarrow 0$;
4.3 $enddist \leftarrow \infty$;
4.4 $maxdist[1..k] \leftarrow \{\infty\}$;
4.5 $Enqueue(Q_p, root, 0, \max_{i=1}^{n}(MaxDist(R_i, root))$;
4.6 **while** Q_p *is not empty and* $end = 0$ **do**
4.7 $\{p, d_{min}(p), d_{max}(p)\} \leftarrow Dequeue(Q_p)$;
4.8 **if** $d_{min}(p) > maxdist[k]$ **then**
4.9 $end \leftarrow 1$;
4.10 **else if** p *is a data point* **then**
4.11 $w(p) \leftarrow \sum_{i=1}^{n} MaxDist(R_i, p)$;
4.12 $A \leftarrow A \cup \{p, w(p)\}$;
4.13 **else**
4.14 **for** *each child node* p_c *of* p **do**
4.15 $d_{min}(p_c) \leftarrow \min_{i=1}^{n} MinDist(R_i, p_c)$;
4.16 $d_{max}(p_c) \leftarrow \max_{i=1}^{n} MaxDist(R_i, p_c)$;
4.17 $Enqueue(Q_p, p_c, d_{min}(p_c), d_{max}(p_c))$;
4.18 **if** $d_{max}(p_c) \leq maxdist[k]$ **then**
4.19 $Update(maxdist, d_{max}(p_c))$;
4.20 $enddist \leftarrow$ Compute_enddist($R_1, R_2, \ldots, R_n, A$);
4.21 $Enqueue(Q_p, p, d_{min}(p), d_{max}(p))$;
4.22 $end \leftarrow 0$;
4.23 **while** Q_p *is not empty and* $end = 0$ **do**
4.24 $\{p, d_{min}(p), d_{max}(p)\} \leftarrow Dequeue(Q_p)$;
4.25 **if** $d_{min}(p) > enddist$ **then**
4.26 $end \leftarrow 1$;
4.27 **else if** p *is a data point* **then**
4.28 $w(p) \leftarrow \sum_{i=1}^{n} MaxDist(R_i, p)$;
4.29 $A \leftarrow A \cup \{p, w(p)\}$;
4.30 **else**
4.31 **for** *each child node* p_c *of* p **do**
4.32 $d(p_c) \leftarrow MinDist(M, p_c)$;
4.33 **if** $d(p_c) \leq enddist[k]$ **then**
4.34 $d_{min}(p_c) \leftarrow \min_{i=1}^{n} MinDist(R_i, p_c)$;
4.35 **if** $d_{min}(p_c) \leq enddist$ **then**
4.36 $Enqueue(Q_p, p_c, d_{min}(p_c), \infty)$;
4.37 $A \leftarrow Refine\_Answers(R_1, R_2, \ldots, R_n, k, \mathcal{R}, A)$;
4.38 **return** A;

We use the notations M, $MinDist(q, p)$, $MaxDist(q, p)$, $maxdist[k]$ as defined in Section 6. We define $enddist$ and redefine $d_{min}(p)$ and $d_{max}(p)$ for kGNN queries based on W-RANK as follows.

- $enddist$: the maximum of the maximum distances between query rectangles $\{R_1, R_2, \ldots, R_n\}$ and the data points in A, i.e., the maximum of $\bigvee_i \bigvee_{p_h \in A} \{MaxDist(R_i, p_h)\}$.
- $d_{min}(p)$ ($d_{max}(p)$): the minimum (maximum) distance of p computed from the minimum (maximum) distances between p and all query rectangles, where p represents a data point or a minimum bounding rectangle of an R^*-tree node.

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 $d_{max}(root) = \max_{i=1}^{n}(MaxDist(R_i, root))$. The elements of Q_p are stored in order of their minimum d_{min}s. The search continues until k nearest data points with respect to every point of each rectangle have been dequeued from Q_p. To preserve a user's privacy while finding the actual GNNs from A through the group of users (the private filter technique discussed in Section 7.2), the weight of each data point p in A is initialized as the sum of the maximum distances of p to the rectangles (Line 4.11). Note that similar to the algorithm for evaluating kGNN queries with respect to a set of rectangles for the aggregate function SUM and MAX, Lemma 1 is used in the algorithm to check the termination condition of the search for k nearest data points with respect to the set of rectangles. After k NNs with respect to every point of each rectangle have been found, the algorithm computes $enddist$ (Line 4.20) and finds all data points whose d_{min}s are less than or equal to $enddist$ (Lines 4.21-4.36). The extension of search for the data points until $enddist$ ensures that A includes k GNNs based on W-RANK with respect to the set of rectangles.

Without loss of generality, we can provide an example that shows why it is required to extend the search until $enddist$ in order to ensure that k GNNs based on W-RANK with respect to every point set within the rectangles are included in A. Let R_1 and R_2 be two rectangles and r_1 and r_2 be two points in R_1 and R_2, respectively. Assume that for $k = 1$, the data points p_1 and p_2 are in A, where p_1 is the 1^{st} and 5^{th} NN for r_1 and r_2, respectively and p_2 is the 4^{th} and 1^{st} NN for r_1 and r_2, respectively. Let p_3 be another data point that is not in A and is the 3^{rd} NN for both r_1 and r_2. For the ratio between two consecutive weights $rt = 2$, i.e., $w_{i+1} = 2 \times w_i$, p_3 has a smaller weight than the weights of p_1 and p_2, i.e., p_3 is the GNN based on W-RANK with respect to points r_1 and r_2. Thus, we observe that simply finding k NNs for r_1 and r_2 cannot ensure that the actual GNN based on W-RANK, i.e., p_3, is included in A. To address the above mentioned scenario, the algorithm extends the search until $enddist$ and includes all data points that have higher ranks (e.g., rank 1 is higher than rank 2) with respect to a point in the rectangles than the ranks of the data points already included in A.

The second part of Algorithm 4 is to prune the data points from A that cannot be part of the candidate answer based on the weight with respect to the set of rectangles. The function $Refine\_Answers$ (Line 4.37) prunes the data set. To compute the weight of a data point with respect to a set of rectangles, we need to know the rank of that data point with respect to each individual rectangle, which is not straightforward to compute. Since a user can be located anywhere within her rectangle, the rank of a data point with respect to a rectangle is a range instead of a fixed rank, which is defined with the minimum and maximum values. Therefore, the weight of a data point with respect to a set of rectangles is also computed as a range with the minimum and maximum bounds.

Algorithm 5 shows the steps of the function $Refine\_Answers$. The first step is to compute the minimum and maximum ranks of the data points for each individual rectangle, which is done with the function $Compute\_Ranks$ (Lines 5.1-5.2). Algorithm 6 shows the steps for the function $Compute\_Ranks$ and we discuss the details of $Compute\_Ranks$ in Section 7.3.1. The main notations in $Refine\_Answers$ and $Compute\_Ranks$ are summarized as follows:

- $rank_{min}(R_i, p_h)$ ($rank_{max}(R_i, p_h)$): the minimum (maximum) rank of a data point p_h with respect to a rectangle R_i.
- $w_{min}(p_h)$ ($w_{max}(p_h)$): the minimum (maximum) weight of a data point p_h with respect to a set of rectangles $\{R_1, R_2, \ldots, R_n\}$.
- $maxw_k$: the k^{th} smallest maximum weight of computed $w_{max}(p_h)$s.

Algorithm 5 Refine_Answers($\{R_1, R_2, \ldots, R_n\}, k, \mathcal{R}, A$)

Input : A set of rectangles $\{R_1, R_2, \ldots, R_n\}$, the number of required GNNs k,a rule $\mathcal{R}$ to compute weights, and the answer set $A := \cup_h\{p_h, w(p_h)\}$.
Output: Updated answer set $A := \cup_h\{(p_h, w(p_h))\}$.

```
5.1  for each rectangle R_i do
5.2      ∪_h{rank_min(R_i, p_h), rank_max(R_i, p_h)} ← Compute_Ranks(R_i, A);
5.3  for each data point p_h ∈ A do
5.4      w_min(p_h) ← 0;
5.5      w_max(p_h) ← 0;
5.6      for each rectangle R_i do
5.7          w_min(p_h) ← w_min(p_h) + w_{rank_min(R_i,p_h)};
5.8          w_max(p_h) ← w_max(p_h) + w_{rank_max(R_i,p_h)};
5.9  maxw_k ⟵ kMax(∪_h{w_max(p_h)});
5.10 for each data point p_h ∈ A do
5.11     if w_min(p_h) > maxw_k then
5.12         remove {p_h} from A;
5.13 return A;
```

After having the minimum and maximum ranks of the data points for all rectangles, the algorithm computes the minimum and maximum weights of each data point in A (Lines 5.3-5.8). Then the algorithm finds the k^{th} smallest maximum weight, $maxw_k$, using the function $kMax$. The next step is to prune the data points from A: the data points whose minimum weight is greater than $maxw_k$ are removed from A since they can never be one of the candidate answers for k GNNs based on W-RANK.

Next, we show how the function $Compute\_Ranks$ (Algorithm 6) finds the minimum and maximum ranks of the data points with respect to a rectangle.

7.3.1 Computing minimum and maximum ranks

We first give the basic idea of computing the minimum and maximum ranks of the data points with respect to a rectangle and then we will discuss the details of the algorithm with an example. To assign the minimum rank, the function $Compute\_Ranks$ divides the data points into groups, where each group of data points have the same minimum rank with respect to a rectangle. The group with the minimum rank, say r, consists of those data

points whose minimum distances from the rectangle are less than or equal to $maxdist[r]$ (i.e., the r^{th} smallest maximum distance of the data points from the rectangle) and greater than $maxdist[r-1]$. Since $maxdist[r-1]$ is less than the minimum distance of any data point in the group with the minimum rank r, none of the data point in the group with the minimum rank r can have a rank lower than r.

On the other hand, to compute the maximum rank of a data point p_h with respect to a rectangle, the function *Compute_Ranks* counts the number of other data points whose minimum distances from the rectangle are less than or equal to the maximum distance of p_h from the rectangle. This is because those data points may have higher rank than the rank of p_h. Since p_h also has a rank, the maximum rank of p_h with respect to the rectangle is determined as one more than the counted number.

Algorithm 6 Compute_Ranks(R_i, A)

Input : A rectangle R_i, and the answer set $A := \cup_h\{p_h, w(p_h)\}$.
Output: The set of ranks $\cup_h\{(rank_{min}(R_i, p_h), rank_{max}(R_i, p_h))\}$.

6.1 compute L_{min_i} and L_{max_i};
6.2 $minrank \leftarrow 1$;
6.3 $dist \leftarrow \infty$;
6.4 $count_1 \leftarrow 0$;
6.5 $count_2 \leftarrow 0$;
6.6 **for** $t \leftarrow 1$ **to** $|A|$ **do**
6.7 **if** $MinDist(R_i, p_{L_{min_i}[t]}) \leq dist$ **then**
6.8 **if** $MaxDist(R_i, p_{L_{min_i}[t]}) \leq dist$ **then**
6.9 $dist \leftarrow MaxDist(R_i, p_{L_{min_i}[t]})$;
6.10 **else**
6.11 $minrank \leftarrow minrank + 1$;
6.12 $dist \leftarrow MaxDist(R_i, p_{L_{min_i}[t]})$;
6.13 $rank_{min}(R_i, p_{L_{min_i}[t]}) \leftarrow minrank$;
6.14 $count_1 \leftarrow count_1 + 1$;
6.15 **for** $s \leftarrow (t+1)$ **to** $|A|$ **do**
6.16 **if** $MinDist(R_i, p_{L_{max_i}[s]}) \leq MaxDist(R_i, p_{L_{max_i}[t]})$ **then**
6.17 $count_2 \leftarrow count_2 + 1$;
6.18 $rank_{max}(R_i, p_{L_{max_i}[t]}) \leftarrow count_1 + count_2$;
6.19 $count_2 \leftarrow 0$;
6.20 **return** $\cup_h\{(rank_{min}(R_i, p_h), rank_{max}(R_i, p_h))\}$;

The details of the working procedure of the function *Compute_Ranks* is as follows. To make the computation faster, the algorithm first computes two arrays L_{min_i} and L_{max_i} of size $|A|$ for computing the minimum and maximum ranks of the data points with respect to a rectangle R_i, respectively (Line 6.1). Each entry $L_{min_i}[t]$ ($L_{max_i}[t]$) corresponds to the value of the index h of a data point p_h that has the t^{th} smallest minimum (maximum) distance from R_i. Tables 8 and 9 (Columns 1-4) show examples of L_{min_1} and L_{max_1} for a rectangle R_1 and answer set $A : \{p_1, p_2, \ldots, p_9\}$. Then the algorithm initializes variables $minrank$, $dist$, $count_1$, and $count_2$ (Lines 6.2-6.5), where the first two variables are used to compute the minimum rank of a data point and the latter ones are used for the maximum rank of the data point.

Table 8 The minimum ranks of the data points in A with respect to R_1

t	$L_{min_1}[t]$	$p_{L_{min_1}[t]}$	$MinDist$	$MaxDist$	$rank_{min}$
1	3	p_3	7	16	1
2	4	p_4	9	20	1
3	1	p_1	10	15	1
4	7	p_7	12	21	1
5	5	p_5	17	27	2
6	6	p_6	18	40	2
7	2	p_2	22	36	2
8	9	p_9	35	50	3
9	8	p_8	37	48	3

The algorithm scans through the data points in A in order of their minimum distances from the rectangle R_i and computes the minimum ranks for them (Lines 6.7-6.13). As mentioned earlier, the data points that are in the same group are assigned the same minimum rank. The variable $minrank$ is used to keep track of the rank that should be assigned as the minimum rank to the scanned data point. The variable $dist$ represents the smallest maximum distance of already scanned data points from R_i for the current group. For each data point $p_{L_{min_i}[t]}$, $dist$ is checked to determine whether the group has been changed. If $MinDist(R_i, p_{L_{min_i}[t]})$ is less than or equal to $dist$ then the group of $p_{L_{min_i}[t]}$ remains the same with that of the previous data point $p_{L_{min_i}[t-1]}$. Otherwise the group is changed, $minrank$ is incremented by 1, and $dist$ is updated to $MaxDist(R_i, p_{L_{min_i}[t]})$. Table 8 shows the minimum rank of $\{p_1, p_2, \ldots, p_9\}$ with respect to the rectangle R_1. For example, in Table 8, we observe that the group of data points $\{p_3, p_4, p_1, p_7\}$ have the minimum rank 1 and the value of $dist$ is 15 for this group. The new group starts from the data point p_5 as $MinDist(R_1, p_5) > 15$.

The algorithm scans through the data points in A in order of their maximum distances from the rectangle R_i and computes the maximum ranks for them (Lines 6.14-6.17). The maximum rank of a scanned data point is computed as the summation of $count_1$ and $count_2$

Table 9 The maximum ranks of the data points in A with respect to R_1

t	$L_{max_1}[t]$	$p_{L_{max_1}[t]}$	$MaxDist$	$MinDist$	$rank_{max}$
1	1	p_1	15	10	4
2	3	p_3	16	7	4
3	4	p_4	20	9	6
4	7	p_7	21	12	6
5	5	p_5	27	17	7
6	2	p_2	36	22	8
7	6	p_6	40	18	9
8	8	p_8	48	37	9
9	9	p_9	50	35	9

(Line 6.18). The variables $count_1$ and $count_2$ are used to track the number of data points that have minimum distances from the rectangle less than or equal to $MaxDist(R_i, p_{L_{max_i}[t]})$. The variable $count_1$ represents the number of the data points whose maximum ranks have been assigned and the data point $p_{L_{max_i}[t]}$ whose maximum rank is currently considered to be computed. Note that the data points whose maximum ranks have been assigned have less or equal maximum distances from the rectangle than $MaxDist(R_i, p_{L_{max_i}[t]})$. On the other hand, $count_2$ counts the number of the data points whose maximum rank has not yet been assigned and the minimum distances of these data points from the rectangle is less than or equal to $MaxDist(R_i, p_{L_{max_i}[t]})$. Table 9 shows the maximum rank of $\{p_1, p_2, \ldots, p_9\}$ with respect to the rectangle R_1. As an example, from the 3^{rd} row of Table 9 we observe that the maximum rank of p_4 is 3. In this case $count_1$ is 3 considering $\{p_1, p_3, p_4\}$ and $count_2$ is 3 considering $\{p_7, p_5, p_6\}$. The minimum distances of $\{p_1, p_3, p_4, p_7, p_5, p_6\}$ are {10, 7, 9, 12, 17, 18}, respectively, which are less than $MaxDist(R_1, p_4)$ (i.e., 20).

7.3.2 Proof of correctness

The following theorem proves the correctness of algorithm REGION_kGNN_W-RANK.

Theorem 1 *If k is the number of required GNNs 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* W-RANK *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 W-RANK 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) p' is not included in A in the first part of the algorithm (Lines 4.1-4.36), or (ii) p' has been pruned in the second part of the algorithm (Line 4.37).

If p' is not included in A in the first part of the algorithm then the minimum distance of p' from the set of rectangles is greater than $enddist$, which means that there are at least k data points whose maximum distances with respect to any rectangle are lower than the minimum distance of p' for that rectangle. Thus, p' cannot have the j^{th} minimum ($1 \leq j \leq k$) value for W-RANK, which contradicts the assumption.

If p' has been pruned in the second part of the algorithm then the minimum weight of p' is greater than the k^{th} smallest maximum weight of the data points in A, which again means p' cannot have the j^{th} minimum ($1 \leq j \leq k$) value for W-RANK and contradicts the assumption. □

7.4 Incremental pruning private filter and privacy attacks

Incremental private pruning filter for kGNN queries based on W-RANK, W-RANK_IPPF, requires that users have the minimum and maximum weights with respect to the set of rectangles (w_{min} and w_{max}) for each candidate data point. If the users have w_{min} and w_{max} then each user can perform local pruning of the data points from the answer set after updating w_{min} and w_{max} of every data point with respect to her actual location. However, in this section, we will show how the update of both w_{min} and w_{max} for the candidate data points in W-RANK_IPPF can put the user's privacy at risk and enable the coordinator or other users in the group to approximate the user's location.

To update $w_{min}(p_h)$ and $w_{max}(p_h)$ for a data point p_h a user u_i computes $w'_{min}(p_h)$ and $w'_{max}(p_h)$ using the following equations:

$$w'_{min}(p_h) = w_{min}(p_h) - w_{rank_{min}(R_i,p_h)} + w_{rank(l_i,p_h)} \tag{13}$$

$$w'_{max}(p_h) = w_{max}(p_h) - w_{rank_{max}(R_i,p_h)} + w_{rank(l_i,p_h)} \tag{14}$$

Then the user assigns $w'_{min}(p_h)$ and $w'_{max}(p_h)$ to $w_{min}(p_h)$ and $w_{max}(p_h)$, respectively. Note that the user can compute $rank_{min}(p_h)$ and $rank_{max}(p_h)$ used in (13) and (14) with Algorithm 6. After the update for all data points, the local pruning steps for each user are same as the pruning steps done by the LSP (Lines 5.10-5.12 in Algorithm 5). From (13) and (14), we observe that the specific pattern of weights can cause privacy threats when users update w_{min} and w_{max} for the data points in W-RANK_IPPF. Without loss of generality consider an example, where the ratio between two consecutive weights is 2, i.e., $w_{i+1} = 2 \times w_i$ and the candidate answer set size is 8; the weights are $\{1, 2, 4, 8, \ldots 256\}$. Suppose, a user increments $w_{min}(p_h)$ of a data point p_h by 6 and decrements $w_{max}(p_h)$ by 8. Then from knowing it, the coordinator or other user can infer that p_h is the 4^{th} nearest data point of the user.

Thus, to protect user privacy our approach does not support W-RANK_IPPF and though the LSP computes w_{min} and w_{max} for each candidate data point, the LSP does not provide both of them to the group.

8 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 the 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 10 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).

We run the experiments on a desktop with a Pentium 2.40 GHz CPU and 2 GByte RAM. We consider 1000 private kGNN queries for each set of experiments, and determine the average experimental results. We randomly generate 1000 point locations that are uniformly

Table 10 Experiment Setup

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

distributed in the total space. Each point p_q in the generated point set 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 the query rectangles for a private kGNN query, we ensure that at least there is one query rectangle that touches each edge of M.

There exists no approach, neither centralized nor decentralized, in the literature that can protect location privacy of users while finding the optimal answer for a GNN query from a large dataset (please see Section 3.1 for details). Though our approach is originally based on a decentralized architecture, it can be also applied for a centralized setting, where the group members disclose their locations to an intermediary entity. The intermediary entity computes the query rectangles for the group members and sends them to the LSP. The LSP evaluates the candidate answers with respect to the rectangles using our proposed algorithms and sends them back to the intermediary entity. The intermediary entity finds the GNNs for the actual locations of the group members and informs them to the group members. Since the decentralized architecture does not reveal a user's location to anyone, even not to other group members, we compare the performance of our algorithms for processing privacy preserving GNN queries by varying different parameters in a decentralized setting.

We present our experimental results of the private filter algorithms and kGNN algorithms with respect to a set of the query rectangles in Section 8.1 and Section 8.2, respectively. In Section 8.3, we show the experimental results for kGNN queries based on weighted-rank.

8.1 Comparison of private filter algorithms

We perform a comparative analysis between our private filter techniques, FPPF and IPPF in terms of computational and communication costs. We measure the time spent by each user in the group for the private filter technique and add them to determine 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 8.1.1 to 8.1.4 and then analyze these results in Section 8.1.5.

8.1.1 Effect of group size

Figure 6a 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 to 2 for the increase of group size from 4 to 16 and then remains constant at 2. For MAX, the time required by IPPF is significantly higher than that of FPPF for a small group size (e.g., 9 times higher for the group size of 4), but with an increase of the group size the time required by FPPF becomes higher than that of IPPF.

We observe in Figure 6b 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 times lower than that of FPPF for SUM and MAX, respectively.

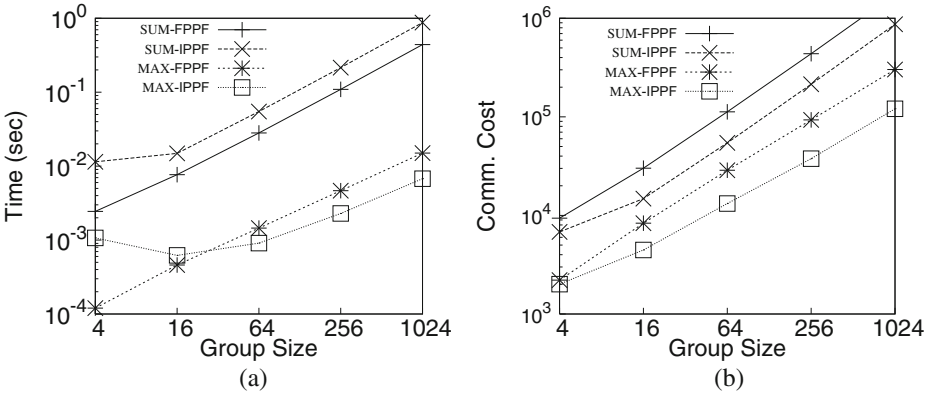


Figure 6 Effect of group size (data set *C*)

8.1.2 Effect of the area of M

We vary the area of M and see in Figure 7a that the time required by IPPF is always 2 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 7b shows that for SUM the communication cost of IPPF is approximately 2 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.

8.1.3 Effect of query rectangle area

Figure 8a 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

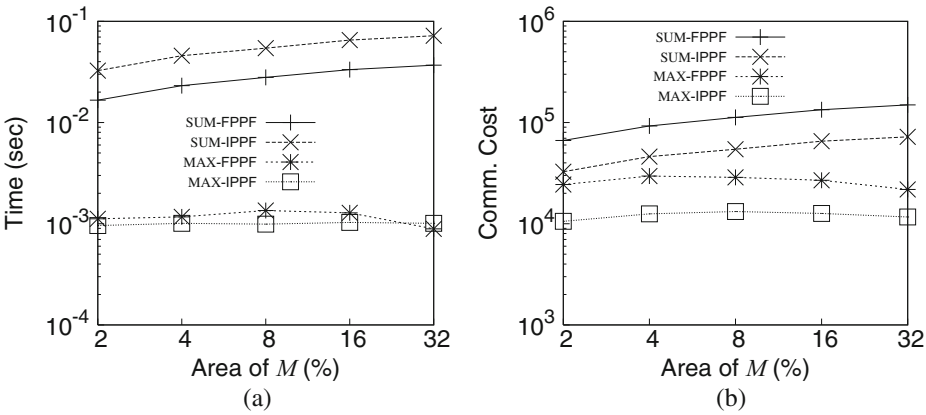


Figure 7 Effect of the area of M (data set *C*)

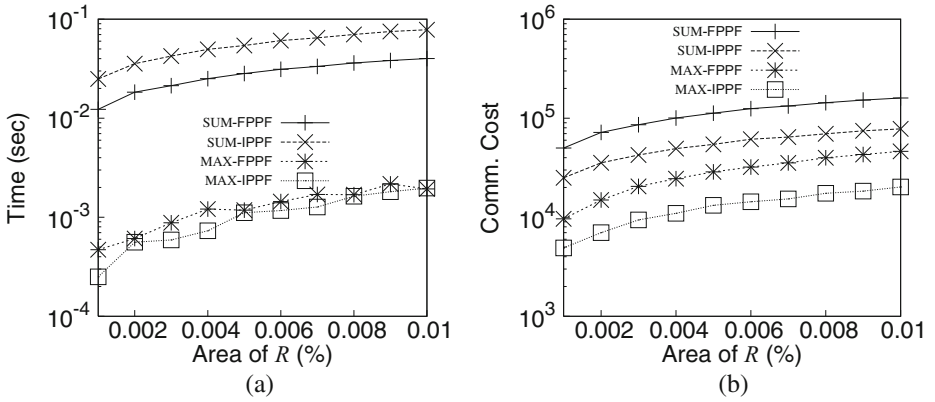


Figure 8 Effect of query rectangle area (data set *C*)

of both IPPF and FPPF vary in a random manner with the increase of the query rectangle area and the time required by IPPF is never higher than that of FPPF.

Figure 8b shows that the communication cost of FPPF is on average 2 and 2.2 times higher than those of IPPF for SUM and MAX, respectively.

8.1.4 Effect of k

The effect of varying k is not significant for SUM as we see in Figure 9a: the times for IPPF and FPPF remain nearly the same for different k and the required time of IPPF is on average 2 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 the query rectangle area and is equal to that of FPPF for $k = 32$.

We observe in Figure 9b 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 for increasing k from 2 to 32.

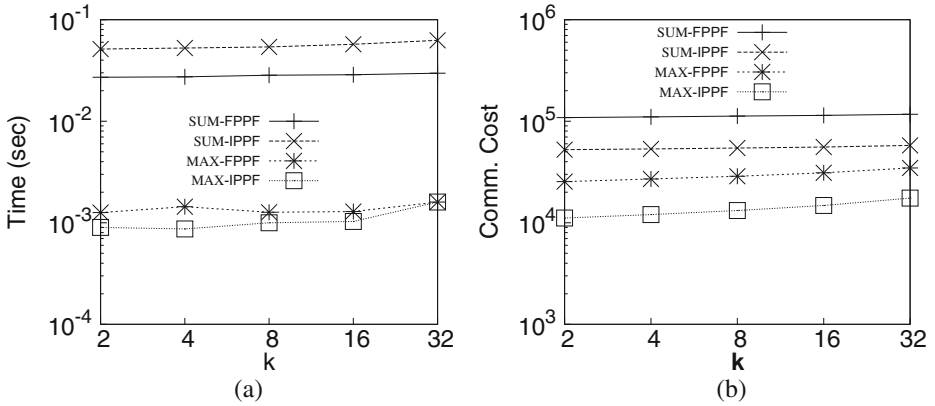


Figure 9 Effect of k (data set *C*)

8.1.5 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.

8.2 Performance of kGNN algorithms with respect to 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.

8.2.1 Effect of group size

In this set of experiments, we vary the group size as 4, 16, 64, 256, and 1024. The processing time increases, and both IOs and the answer set size decrease with the increase of the group size for both SUM and MAX (Figure 10a–f). The reason behind the increase of the processing time with the group size is the increase of the number of distance computations involved to determine an aggregate distance. On the other hand, the reason for the decrease of IOs and the answer set size 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 d_{min} of each data point or R^*-tree node is considered to check if the data point or R^*-tree node can be pruned. Hence, the probability is high that more d_{min}s become larger

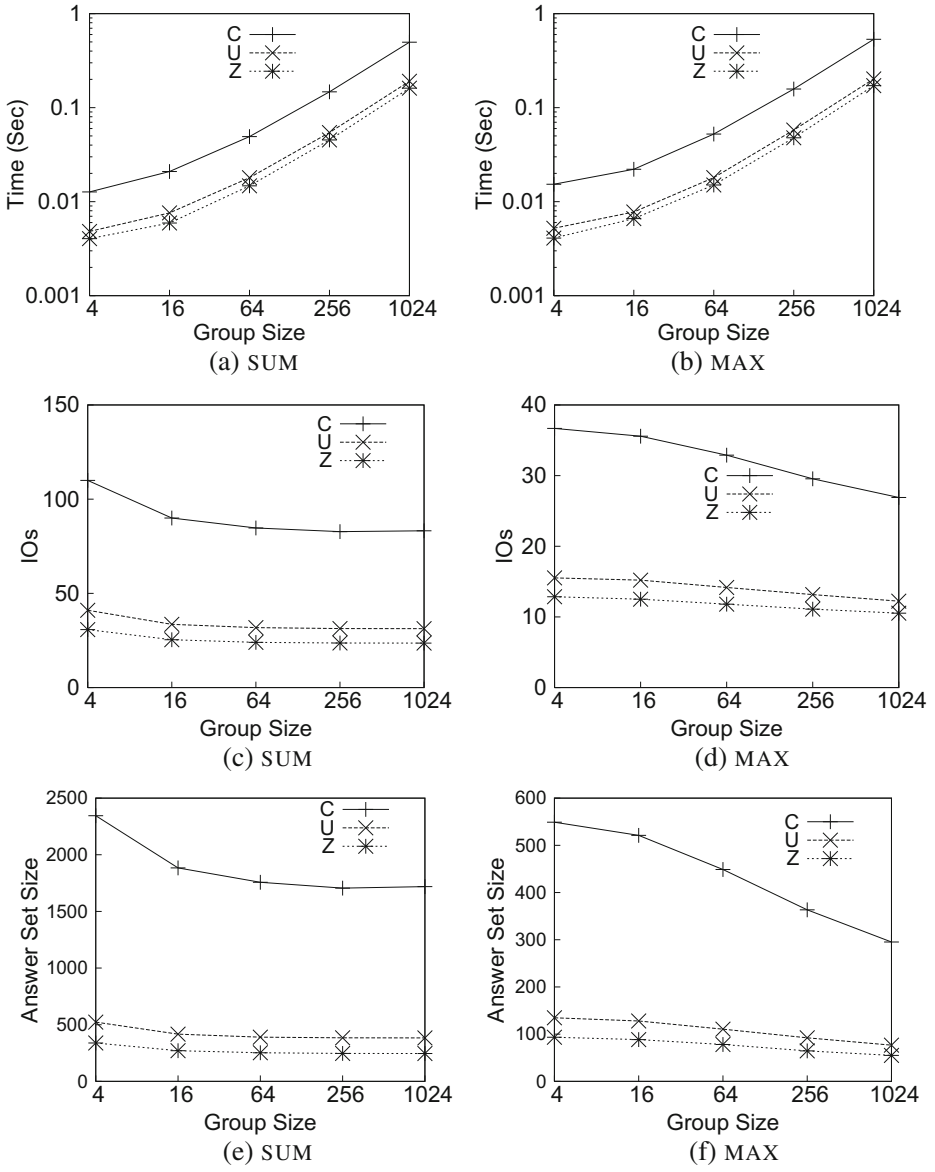


Figure 10 Effect of group size

than $maxdist_k$ with an increased group size and more data points or R^*-tree nodes are pruned.

8.2.2 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. We show the results for the required time in Figure 11.

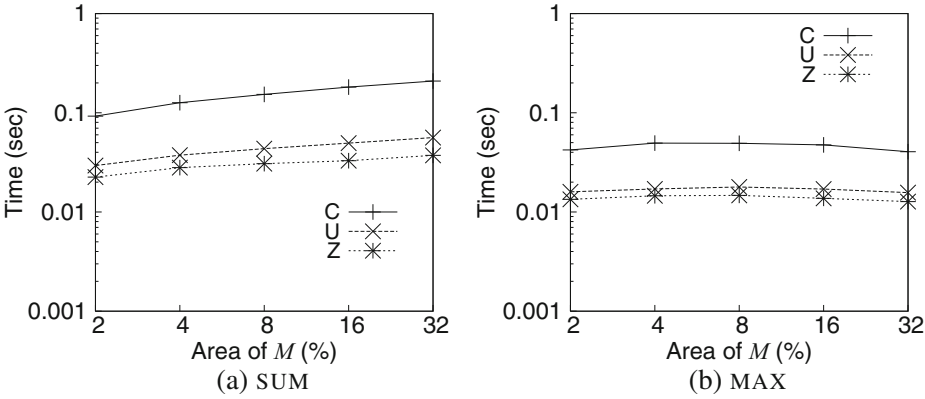


Figure 11 Effect of the area of M

There are two factors that influence the outcome of these experiments. Both d_{min} and d_{max} of the data points or R^*-tree nodes that are outside a smaller M decrease or remain 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 the data points or R^*-tree nodes that are inside of a smaller M decrease or remain 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.

8.2.3 Effect of query rectangle area

In this set of experiments, we vary the query rectangle area and observe that the processing time, IOs, and the answer set size increase with the increase of the query rectangle area. With the increase of query rectangle area, for each data point, d_{min} decreases or remains 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 12).

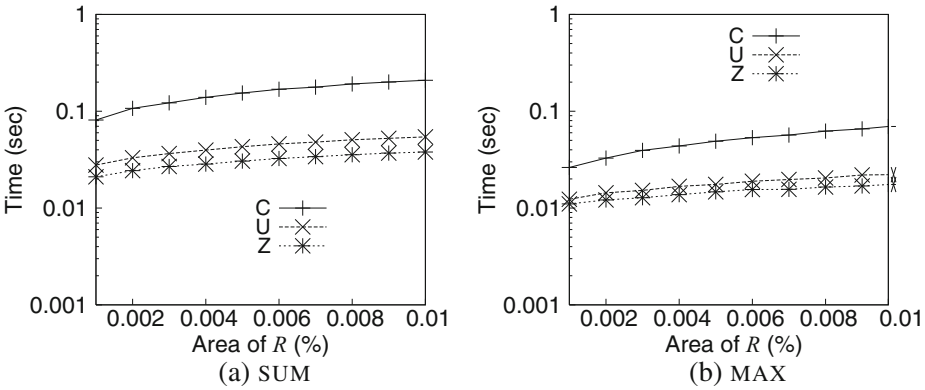


Figure 12 Effect of query rectangle area

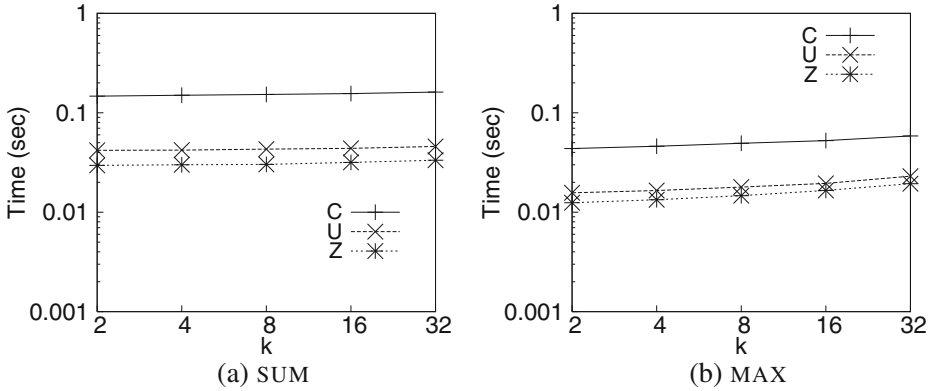


Figure 13 Effect of k

8.2.4 Effect of k

In this set of experiments we vary k and observe that the processing time (Figure 13), IOs, and the answer set size slightly increase with the increase of k, which is expected because a larger value of k increases $maxdist_k$ and less R^*-tree nodes or data points are pruned.

8.2.5 Effect of data set size

In this set of experiments we investigate the effect of the data set size and find that the processing time, IOs, and the answer set size increase for increasing data set sizes and the rate of increase decreases for a larger data set. For example, when we increase the data set size from 5k to 10k the increase ratio of the processing time are 1.5 (SUM) and 1.1 (MAX), whereas when we increase the data set size from 15k to 20k the increase ratio of the processing time are 1.2 (SUM) and 1.1 (MAX) (Figure 14).

For each set of experiments, except the experiments in Sections 8.1.3 and 8.2.3, we also consider the case, where the users of a group have variable privacy levels, i.e., the area of

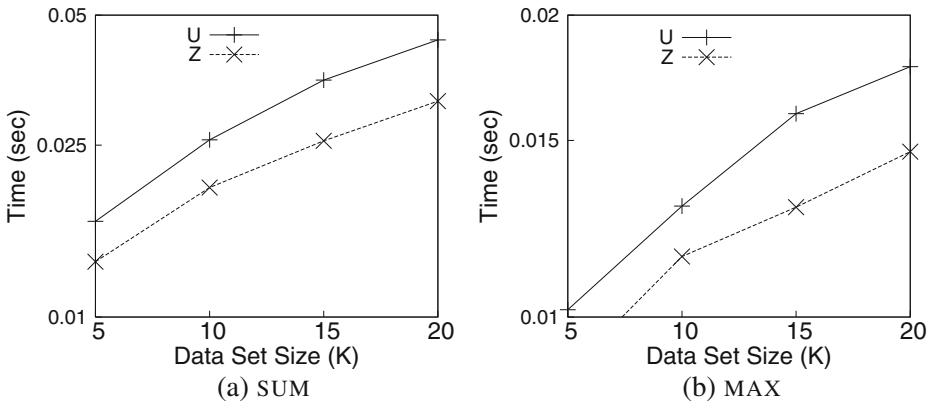


Figure 14 Effect of data set size

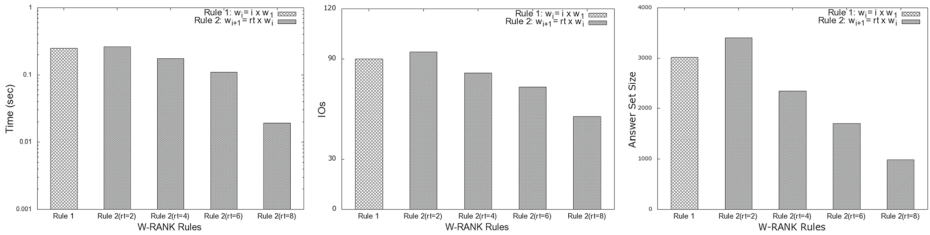


Figure 15 Effect of W-RANK rules: LSP side

the query rectangles are different for a group. We find that the experimental results show similar trends to those for equally-sized query rectangles.

8.2.6 Summary

Our algorithm for private kGNN queries is scalable as it can handle a large group size (up to 1024). Furthermore, the processing cost required by our algorithm slightly increases with the increase of user privacy level, i.e., the area of a query rectangle.

8.3 Private kGNN Queries based on W-RANK

For private kGNN queries based on W-RANK, the group can set the rules to compute the set of weights. We use two rules in our experiments. Rule 1 is $w_i = i \times w_1$, where the ratio between w_1 and w_i is set to i, and w_i is computed as $i \times w_1$. Another rule is $w_{i+1} = rt \times w_i$, where the ratio between two consecutive weights w_{i+1} and w_i is set to rt. In our experiment, we set w_1 as 1 and vary rt as 2, 4, 6 and 8. We set other parameters to default values as shown in Table 10. We run the experiments on a desktop with a Pentium(R) Dual Core 2.30 GHz CPU and 3 GByte RAM. We consider 100 private kGNN queries based on W-RANK and compute the average performance for the following sets of experiments.

Figure 15a–15c show the processing time, IOs, and the answer set size, respectively, for the LSP side algorithm to evaluate kGNN queries based on W-RANK with respect to a set

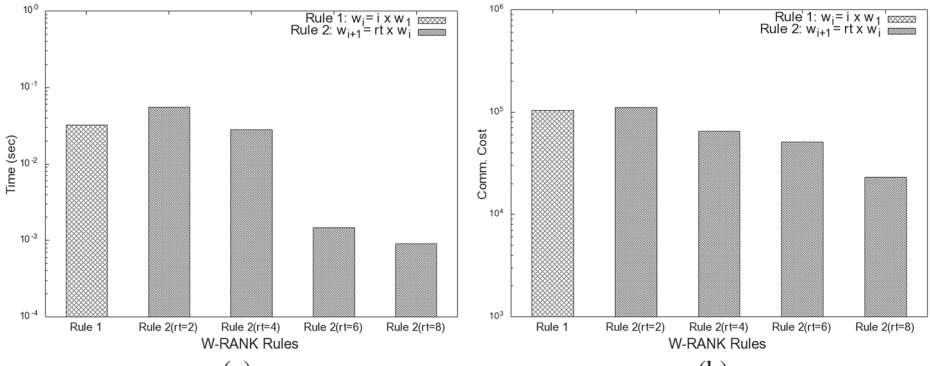


Figure 16 Effect of W-RANK rules: user side

of rectangles. Figure 16a–b show the processing time and communication cost required by final pruning private filter (FPPF) for private kGNN queries based on W-RANK. We observe that both rules show similar performance when the value of r is 2. On the other hand, the performance for Rule 2 improves with the increase of rt, which is expected.

9 Conclusion

The paper presents an efficient solution to process privacy preserving group nearest neighbor (GNN) queries in a decentralized manner. The solution is composed of two components: (i) actual GNN identification from the candidate answers using private filters, and (ii) candidate answer computation with respect to a set of regions with efficient algorithms. Our solution can evaluate GNN queries for three aggregate functions, SUM, MAX and W-RANK, where SUM and MAX minimizes the total and the maximum distance of the group members to the data points, respectively, and W-RANK maximizes the overall preference of the group members for the data points. Extensive experiments for different parameter settings show that our solution is scalable and can ensure high level of user privacy in real-time.

In the future, we intend to investigate the possibility of a privacy preserving solution for GNN queries based on the assumption that all involved entities in processing the queries may corroborate to extract extra information about a user's location. We will explore to what extent secure multi-party computations (e.g., [6]) can be used to enhance our approach.

References

1. Ahmad, S., Kamal, R., Ali, M.E., Qi, J., Scheuermann, P., Tanin, E.: The flexible group spatial keyword query. In: ADC, pp. 3–16 (2017)
2. Ashouri-Talouki, M., Baraani-Dastjerdi, A., Selçuk, A.A.: Glp A cryptographic approach for group location privacy. Comput. Commun. **35**(12), 1527–1533 (2012)
3. Ashouri-Talouki, M., Baraani-Dastjerdi, A., Selcuk, A.A.: The cloaked-centroid protocol: location privacy protection for a group of users of location-based services. Knowl. Inf. Syst. **45**(3), 589–615 (2015)
4. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. SIGMOD Rec. **19**(2), 322–331 (1990)
5. Bettini, C., Mascetti, S., Wang, X.S., Jajodia, S.: Anonymity in location-based services: Towards a general framework. In: MDM, pp. 69–76 (2007)
6. Bickson, D., Dolev, D., Bezman, G., Pinkas, B.: Peer-to-peer secure multi-party numerical computation. In: P2P, pp. 257–266 (2008)
7. Bilogrevic, I., Jadliwala, M., Kalkan, K., Hubaux, J.-P., Aad, I.: Privacy in mobile computing for location-sharing-based services. In: PETS, pp. 77–96 (2011)
8. Chow, C.-Y., Mokbel, M.F., Liu, X.: A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In: GIS, pp. 171–178 (2006)
9. Facebook. http://www.facebook.com
10. Fischer, I., Gotsman, C.: Fast approximation of high-order voronoi diagrams and distance transforms on the gpu. J. Graph. Tools **11**(4), 39–60 (2006)
11. Freudiger, J., Shokri, R., Hubaux, J.: Evaluating the privacy risk of location-based services. In: Financial cryptography and data security, pp. 31–46 (2011)
12. Gedik, B., Liu, L.: Protecting location privacy with personalized k-anonymity: Architecture and algorithms. TMC **7**(1), 1–18 (2008)
13. Ghinita, G., Kalnis, P., Skiadopoulos, S.: Mobihide: A mobile peer-to-peer system for anonymous location-based queries. In: SSTD, pp. 221–238 (2007)
14. Ghinita, G., Kalnis, P., Skiadopoulos, S.: PRIVÉ Anonymous location-based queries in distributed mobile systems. In: WWW, pp. 371–389 (2007)
15. Ghinita, G., Kalnis, P., Khoshgozaran, A., Shahabi, C., Tan, K.-L.: Private queries in location based services: anonymizers are not necessary. In: SIGMOD, pp. 121–132 (2008)

16. Ghinita, G., Damiani, M.L., Silvestri, C., Bertino, E.: Preventing velocity-based linkage attacks in location-aware applications. In: GIS, pp. 246–255 (2009)
17. Google+. http://plus.google.com
18. Gruteser, M., Grunwald, D.: Anonymous usage of location-based services through spatial and temporal cloaking. In: MobiSys, pp. 31–42 (2003)
19. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
20. Hashem, T., Kulik, L.: Safeguarding location privacy in wireless ad-hoc networks. In: Ubicomp, pp. 372–390 (2007)
21. Hashem, T., Kulik, L., Zhang, R.: Privacy preserving group nearest neighbor queries. In: EDBT, pp. 489–500 (2010)
22. Hashem, T., Kulik, L.: "Don't trust anyone": Privacy protection for location-based services. Perv. Mob. Comput. **7**, 44–59 (2011)
23. Hashem, T., Ali, M.E., Kulik, L., Tanin, E., Quattrone, A.: Protecting privacy for group nearest neighbor queries with crowdsourced data and computing. In: UbiComp, pp. 559–562 (2013)
24. Hashem, T., Hashem, T., Ali, M.E., Kulik, L.: Group trip planning queries in spatial databases. In: SSTD, pp. 259–276 (2013)
25. Hashem, T., Kulik, L., Zhang, R.: Countering overlapping rectangle privacy attack for moving knn queries. Inf. Syst. **38**(3), 430–453 (2013)
26. Hashem, T., Barua, S., Ali, M.E., Kulik, L., Tanin, E.: Efficient computation of trips with friends and families. In: CIKM, pp. 931–940 (2015)
27. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: SSD, pp. 83–95 (1995)
28. Hu, H., Lee, D.L.: Range nearest-neighbor query. TKDE **18**(1), 78–91 (2006)
29. Hu, H., Xu, J.: Non-exposure location anonymity. In: ICDE, pp. 1120–1131 (2009)
30. Huang, Y., Vishwanathan, R.: Privacy preserving group nearest neighbour queries in location-based services using cryptographic techniques. In: GLOBECOM, pp. 1–5 (2010)
31. Huang, J., Peng, M., Wang, H., Cao, J., Gao, W., Zhang, X.: A probabilistic method for emerging topic tracking in microblog stream. World Wide Web **20**(2), 325–350 (2017)
32. Jahan, R., Hashem, T., Barua, S.: Group trip scheduling (GTS) queries in spatial databases. In: EDBT, pp. 390–401 (2017)
33. Khoshgozaran, A., Shahabi, C.: Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In: SSTD, pp. 239–257 (2007)
34. Li, H., Lu, H., Huang, B., Huang, Z.: Two ellipse-based pruning methods for group nearest neighbor queries. In: GIS, pp. 192–199 (2005)
35. Li, F., Yao, B., Kumar, P.: Group enclosing queries. TKDE **23**(10), 1526–1540 (2011)
36. Li, M., Sun, X., Wang, H., Zhang, Y., Zhang, J.: Privacy-aware access control with trust management in Web service. World Wide Web **14**(4), 407–430 (2011)
37. Li, Y., Li, F., Yi, K., Yao, B., Wang, M.: Flexible aggregate similarity search. In: SIGMOD, pp. 1009–1020 (2011)
38. Li, J., Thomsen, J.R., Yiu, M.L., Mamoulis, N.: Efficient notification of meeting points for moving groups via independent safe regions. TKDE **27**(7), 1767–1781 (2015)
39. Loopt. http://www.loopt.com
40. Luo, Y., Chen, H., Furuse, K., Ohbo, N.: Efficient methods in finding aggregate nearest neighbor by projection-based filtering. In: ICCSA, pp. 821–833 (2007)
41. Microsoft. Location & privacy: Where are we headed?, 2011 (accessed September 30, 2017). https://news.microsoft.com/location_and_privacy_where_are_we_headed__web
42. Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: VLDB, pp. 763–774 (2006)
43. Namnandorj, S., Chen, H., Furuse, K., Ohbo, N.: Efficient bounds in finding aggregate nearest neighbors. In: DEXA, pp. 693–700 (2008)
44. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE, p. 301 (2004)
45. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. TODS **30**(2), 529–576 (2005)
46. Peng, M., Zeng, G., Sun, Z., Huang, J., Wang, H., Tian, G.: Personalized app recommendation based on app permissions. World Wide Web (2017)
47. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD, pp. 71–79 (1995)
48. Schlegel, R., Chow, C., Huang, Q., Wong, D.S.: User-defined privacy grid system for continuous location-based services. TMC **14**(10), 2158–2172 (2015)
49. Strassman, M., Collier, C.: Case study: The development of the find friends application. In: Location-Based Services, pp. 27–40 (2004)

50. Sun, X., Wang, H., Li, J., Truta, T.M.: Enhanced p-sensitive k-anonymity models for privacy preserving data publishing. Trans. Data Privacy **1**(2), 53–66 (2008)
51. Wang, H., Zhang, Z., Taleb, T.: Special issue on security and privacy of iot. World Wide Web, 1–6 (2017)
52. Xue, M., Kalnis, P., Pung, H.K.: Location diversity: Enhanced privacy protection in location based services. In: International Symposium on Location and Context Awareness, pp. 70–87 (2009)
53. Yi, X., Paulet, R., Bertino, E., Varadharajan, V.: Practical approximate k nearest neighbor queries with location and query privacy. TKDE **28**(6), 1546–1559 (2016)
54. Yiu, M.L., Jensen, C.S., Huang, X., Lu, H.: Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In: ICDE, pp. 366–375 (2008)
55. Zhang, J., Tao, X., Wang, H.: Outlier detection from large distributed databases. World Wide Web **17**(4), 539–568 (2014)