A Crowd Enabled Approach for Processing Nearest Neighbor and Range Queries in Incomplete Databases with Accuracy Guarantee

Mehnaz Tabassum Mahin[1], Tanzima Hashem[2], Samia Kabir[3]

Department of CSE, Bangladesh University of Engineering and Technology, Dhaka-1000
[1]*mehnaztabassummahin@cse.buet.ac.bd,* [2]*tanzimahashem@cse.buet.ac.bd,*
[3]*samialiya212@gmail.com*

Abstract

With the proliferation of mobile devices and wireless technologies, location based services (LBSs) are becoming popular in smart cities. Two important classes of LBSs are Nearest Neighbor (NN) queries and range queries that provide user information about the locations of a point of interests (POIs) such as hospitals or restaurants. Answers of these queries are more reliable and satisfiable if they come from trustworthy crowd instead of traditional location service providers (LSPs). We introduce an approach to evaluate NN and range queries with crowdsourced data and computation that eliminates the role of an LSP. In our crowdsourced approach, a user evaluates LBSs in a group. It may happen that group members do not have knowledge of all POIs in a certain area. We present efficient algorithms to evaluate queries with accuracy guarantee in incomplete databases. Experiments show that our approach is scalable and incurs less computational overhead.

Keywords: Crowdsourcing, Incomplete Databases, Nearest Neighbor Queries, Range Queries

1. Introduction

Location based services (LBSs) are important components of smart city services [1]. Answers of these location based queries are more reliable and satisfiable if they come from trustworthy crowd instead of traditional location service providers (LSPs). In smart cities, the advancement and proliferation of mobile devices and wireless technologies have enabled their citizens to remain connected from any place at any time. In near future, we envision that LBSs will be offered with crowdsourced data and computation without involving an LSP. *Nearest Neighbor* (NN) queries [2, 3] and *range* queries [4, 5] are two important classes of LBSs. A NN query provides a user with the nearest point of interest (POI) such as a hospital, a restaurant, a shopping centre or a movie theatre. A k Nearest Neighbor (kNN) query is an extension of a NN query that returns

k nearest POIs from a user's query location. A range query returns all POIs within a certain area. In this paper, we present a novel approach for efficient processing of kNN and range queries in a crowdsourced manner.

In our crowdsource based approach, a user forms a group with friends and families, and evaluates LBSs in co-operation with group members. Traditional techniques [3, 5, 6] to process kNN and range queries assume that POI data are stored on an LSP's database. Users who want to evaluate kNN or range queries can send locations or desired areas, respectively, to the LSP and the LSP returns the resultant set of POIs to users. However, these techniques suffer from the following limitations: (i) a user may not feel comfortable to share locations with an unknown LSP as the LSP can infer a user's habit, health and preference from the revealed user's location [7, 8], (ii) an LSP may be biased for some POIs and return those POIs as answers, which are not the actual nearest ones [9, 10], (iii) since an LSP is not aware of a user's personal choice and preference for POIs, the returned POIs may not satisfy the user's purpose. To overcome these limitations, users may prefer to involve their similar-minded and trusted peers instead of an unknown and impersonal LSP for processing LBSs. The incentive behind joining a group could be the mutual benefits. It is expected that if a user participates in the query evaluation process of a peer, the peer also participates in the query evaluation process of the user. We introduce a technique using crowdsourced data and computation that completely eliminates the role of an LSP while processing LBSs. Efficient processing of kNN queries [2, 3] and range queries [4, 5] on an LSP's database have been extensively studied in the literature. To the best of our knowledge, this work is the first to process kNN and range queries in a crowdsourced manner.

A major challenge of processing kNN and range queries in a crowdsourced manner is that the group members may not have complete information, i.e., they may not know the locations of all POIs in a certain area. Thus, kNN and range queries have to be evaluated in incomplete distributed databases that may result in a slight decrease in the accuracy of the query answer. In most of the cases, users are happy to accept a slightly lower accuracy if in return they can protect their location privacy from an untrusted LSP and receive query answers that match with their choices and preferences. We develop techniques to measure the confidence level [11] (i.e., accuracy) of the query answer based on aggregate knowledge of group members. In this paper, we propose efficient algorithms to determine the answers for kNN and range queries in incomplete databases with accuracy guarantee.

Users normally know the locations of POIs in the areas where they live or work, and like other crowdsource based approaches [12] we assume that users store their POI knowledge on their mobile devices. This POI knowledge can be used to evaluate answers of kNN and range queries from their friends and families, who do not know the locations of POIs in a certain area. In addition to POI information, we exploit geometric properties to derive the space knowledge of users and store them to compute the confidence level of the evaluated query answers. For example, a user may know the location of the nearest hospital from her home. From this knowledge, we can derive that there is no other hospital in

the circular area where the centre of the circular area is located at the location of the user's home and the radius of the circular area is equal to the distance between the hospital and the user's home. Identifying the known regions in the total space enables us to determine the accuracy of the query answers. We use spatial indexing techniques like R-trees and quad-trees to store POI and space data, respectively, and develop efficient algorithms to evaluate kNN and range queries, independently on the group member's device.

Recently, a crowsource based approach [12] has been developed for processing group nearest neighbor (GNN) queries on a small scale POI set. A GNN query returns a POI that has the minimum aggregate distance with respect to the locations of group members. This work assumes that each group member provides a fixed number of POIs and the GNN query is evaluated on the aggregated POI-set. On the other hand, we evaluate kNN and range queries on a large scale POI-set by considering every POI stored on each group member's local database. The focus of [12] is to hide locations of users from other group members, which is different from our work. Our work considers the incompleteness of the POI data and evaluates kNN and range queries with accuracy guarantee.

In summary, the contributions of our work are as follows:

- We present a novel idea to evaluate LBSs with crowdsourced data and computation. Our approach eliminates the role of an LSP.
- We propose efficient algorithms to evaluate kNN and range queries efficiently in incomplete distributed databases.
- We employ geometric properties to store space information along with POI information on users' mobile devices. We compute the confidence level of an answer using space information and provide the answer to the user with accuracy guarantee.
- We present extensive experiments in a simulated environment using a real dataset to validate the applicability of our approach. We consider both sequential and parallel processing of kNN and range queries, and perform a comparative analysis between sequential and parallel processing in terms of processing time and communication cost.

The remainder of the paper is organized as follows. Section 2 gives an overview of our system. In Section 3, we discuss the crowdsource based data collection and indexing techniques used in our system. In Section 4, we present our approach to process kNN and range queries with crowdsourced data and computation. In Section 5, we show experimental setup and result using real datasets. In Section 6, we discuss a rating system that can ensure the trustworthiness and reputation of group members who participate in the evaluation process. Section 7 discusses the related works in the literature. Section 8 concludes the paper with future research directions.

2. System Overview

In our system, we assume that a user, willing to access LBSs in a crowdsourced manner, forms a group with friends and relatives (see Section 4.1 for details). Peers who accept the invitation of a user to form a group, agree to participate in the query evaluation process of the user based on their stored knowledge on the mobile devices. Note that forming a group does not mean that a user gets access to the local databases of group members. Group formation only permits a user to request group members for evaluating a kNN or range query based on the user's provided location. Each group member evaluates the query on her own database and returns POI and space information related to the received query. Group members have rights to leave the group anytime if they do not want to share their knowledge with others.

(a) Parallel processing **(b)** Sequential processing

Figure 1: Communication flow of the system

At the time of requesting a kNN or range query, considering the query location, a user may select a subset of members from the formed group based on different criteria such as area knowledge of members (e.g., the probability is high that a member is aware of the POIs of a suburb, where she lives). Figure 1 shows the generalized overview of our proposed system. The query evaluation process can be either parallel or sequential that trades off between processing time and communication cost.

Parallel Processing: In the parallel processing (Figure 1(a)), a user sends the location and a query (kNN or range) to the selected group members. Each group member independently evaluates the query on the local database and returns POI and space information to the query requestor (i.e., the user who initiates the query). After receiving POI and space information from group members, the query requestor evaluates the final answer from the aggregate information and computes the confidence level for the query answer. The advantage of using parallel processing is its scalability in terms of the number of group members involved in processing the query. Since computations of group members are performed in parallel, the processing time does not increase with the increase of the number of participants in the query evaluation process. On the other hand, the parallel processing may cause retrieval of same POI and space information from multiple group members and increase the communication overhead.

Sequential Processing: To reduce the communication overhead, the sequential processing can be used. In the sequential processing (Figure 1(b)), the query requestor selects a member from the group and sends the IDs of all members with visited flags, who are involved in the query evaluation process, in addition to the query (kNN or range). Initially, all visited flags are set to false. The member who receives the information, evaluates the query on her local database, appends the retrieved POI and space information from the local database to the received message (which contains IDs, flags and the query) and marks the visited flag associated with her ID as true. Then the member selects another member from the IDs whose visited flags are false and forwards the message to the selected member. The member who receives the information next, evaluates the query on her local database, and checks whether the retrieved information (POI and space data) from her local database can improve the query answer already in the received message. If yes, then the member updates the message. Then the member marks the visited flag associated with her ID as true, and forwards the updated message to the next selected member. The process continues until all visited flags become true. At the end, the message is forwarded to the query requestor. The query requestor evaluates the final answer from the aggregated information and computes the confidence level for the query answer.

The advantage of using sequential processing is its reduced communication overhead as the query requestor does not receive same POI and space information from multiple members. However, the reduced communication cost comes in return of increasing the query processing time. The processing time increases with the increase of the number of group members involved in the query evaluation process. Thus, a user may select the sequential or the parallel processing depending on her requirement. For example, if a user want to evaluate the query in a minimum time, the user may prefer the parallel processing, whereas if a user has communication bandwidth constraint, the user may prefer the sequential processing.

3. Data Collection and Indexing

Our approach evaluates kNN and range queries based on the data from crowd. Like other crowdsourced approaches [12], we envision that users store POI knowledge on their local devices and later use these information to evaluate LBSs. For example, when a user visits a POI, she may input the location of the POI using her GPS-enabled mobile device. In addition to POI location and type, our approach assumes that users also store space knowledge to facilitate the computation of the confidence level [11] of a query answer. For example, a user may know the location of the nearest pharmacy from her home or may know the locations of all hospitals in a suburb. These information can reveal information about the known region to the user in the total space.

Exploring all POIs while evaluating a kNN or range query would incur high computational overhead. To make the search faster, we index POI data using in memory R-tree [13] on the user's local device and call it as a *POI-tree*. On

the other hand, we exploit geometric properties to identify the known region to a user in the total space (i.e., space knowledge) and store on the memory of the user's local device using quad-trees [14]. We call it as a *space-tree*. Figure 2 shows two examples of deriving known region from the POI knowledge of users.

(a) p is the NN from q **(b)** p is the NN along the path from s to d

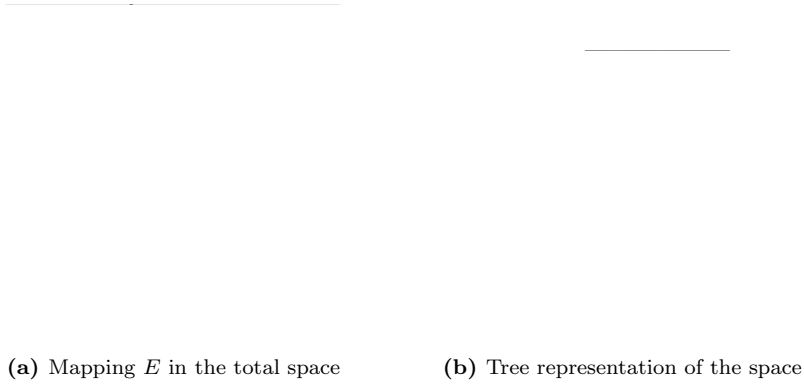
Figure 2: Derived bounded regions

Circular Region: Let a user located at q know that the nearest train station from q is located at p. From this knowledge, we can derive a known circular region that has the center at q and the radius equal to the distance between q and p (Figure 2(a)). Since p is the nearest train station with respect to q, there is no other train stations within the circular region. Thus, this circular region can be stored as the known region in the space-tree and be used in computing the confidence level of the query answer.

Elliptical Region: Let a user know that p is the train station that minimizes the distance of a path starting from a source location s to a destination d via a train station. From this knowledge, we can derive a known elliptical region that has foci at s and d, and the major axis equal to the distance between s and d via p (Figure 2(b)). Since p is the train station that minimizes the travel distance with respect to s and d, there is no other train stations within the elliptical region. Thus, this elliptical region can be stored as the known region in the space-tree and be used in computing the confidence level of the query answer.

Indexing Space Knowledge: A space-tree (i.e., a quad-tree) is a hierarchical data structure, where the root node of the space-tree represents the total space, and is recursively decomposed into four quadrants. The decomposition of a quadrant stops if it is identified as *known* or *unknown* or the depth of the space-tree reaches its maximum limit. A quadrant is known and its associated tree node is marked with 1, if the area represented with the node is included in the known region. A quadrant is unknown and its associated tree node is marked with 0, if the area represented with the node does not intersect with the known region. If an area is partially covered by the known region, we mark the node with 0, and the area is further divided into four quadrants.

For the ease of presentation, we assume that initially a user only knows an elliptical area (E) as the known region. Figure 3(a) shows that how E is mapped in a space-tree, and Figure 3(b) shows the quad-tree representation of the total space. Let the maximum depth of the space-tree be 4. To build the space-tree, the root node is first decomposed into four quadrants SW, SE, NE, and NW. Since SE, NE, and NW are unknown, they are not further decomposed. Node



(a) Mapping E in the total space **(b)** Tree representation of the space

Figure 3: Building the space-tree

SW is decomposed into R_1, R_2, R_3, and R_4. Since R_1 and R_4 are partially covered, they are decomposed again, and the decomposition continues until the tree depth reaches 4. Nodes R_{41} are R_{42} are marked with 1, and not decomposed again because they are included in the known regions. Figure 4 shows that how NE node of the space-tree is further updated when the user becomes aware of a new circular known region (C).

(a) Mapping C in the total space **(b)** Updated part of the space-tree

Figure 4: Updated space-tree

We use hierarchical indexing methods like R-trees and quad-trees because they allow us to search efficiently by pruning irrelevant parts of the trees that are guaranteed not include the query answers during the query evaluation phase (see Section 4.2.1 for detail). Specifically, we use R-trees for indexing POIs because it has been shown in the literature [13] as one of the most efficient methods for processing nearest neighbor queries. On the other hand, quad-trees suite best for our context of partitioning the space based on the known and unknown areas. Note that our approach is applicable for any other indexing method that suits our context for storing POI and space knowledge.

4. Our Approach

In a crowdsource based approach, users may not have knowledge of all POIs, which may lead to incomplete databases. In this section, we propose efficient algorithms to evaluate kNN and range queries in incomplete databases with accuracy guarantee. Our proposed crowd enabled approach work in two phases for evaluating a kNN or a range query. First phase is to select a set of trustworthy people based on different criteria like area knowledge. We refer this phase as "Group Formation" and describe the details of the group formation step in Section 4.1. After selecting group members, a user can evaluate a kNN or a range query in the second phase. We refer this second phase as "Query Processing" and present in Section 4.2.

4.1. Group Formation

The precondition of any crowdsource based system is to select a potential set of crowd. We refer to this potential set of crowd as a group and the overall procedure as the group formation. Members of this group can share their knowledge with a user who forms the group by participating in the processing of the user's queries. The quality of collected data from the crowd widely depends on the knowledge of the group members. In our system, we consider the group formation as a two-step procedure: (i) primary selection of a set of friends F, and (ii) query time selection of a group $G \subseteq F$, which are discussed in Section 4.1.1 and 4.1.2, respectively. The privacy issues that are considered during the group formation re elaborated in Section 4.1.3.

4.1.1. Primary Selection

We assume that a user is a member of a social network, where members of the network have public profiles describing different parameters like the area where she lives and works, hobbies, and preferences such as food or book choice. In this selection step, a user selects a set of n' users in her network in order to form a primary group F, and sends request to them for forming the primary group. A user may use a profile or behavior similarity matching technique for selecting n' users in her network, which is orthogonal to the scope of this paper and we leave it for our future work. In the similar way, a social network member before accepting a request can check her profile or behavior similarity with the requestor. When a social network member accepts a user's request, the member becomes part of the user's primary group F and agrees to participate in processing user queries. Both the member and the user provide access to each other in more detail information of their profile. The detail information helps a user to form an appropriate group by selecting members from F for evaluating a specific query. A member in F can leave any time if the member does not want to continue with a user.

4.1.2. Query Time Selection

In this selection step, a query requestor selects n members from the primary group F in order to form the group G to whom she wants to send a query,

where $n \leq n'$. While forming G, a query requestor may consider the available information from members' profiles and the area knowledge of members in F. For example, if a query requestor wants to know about POIs of a specific suburb using a range query, the query requestor may prefer to select a member from F, who lives or works in the surrounding area, assuming that she may have knowledge about the area. A member in F, who lives far away from the specific suburb and has never visited the suburb included in a range query, is less preferable for forming G. Furthermore, if the requestor is benefited by the query result or effective response from a member of F, she may give more priority to that member while forming G for the next time.

4.1.3. Privacy Issues

When a member u_i accepts the request of a query requestor and becomes a member of group F, both query requestor and u_i can access the current profile and regular updates of each other. Note that other members of F do not have access to u_i's profile, and u_i also does not have access to the profiles of other members for privacy reasons. One may raise concern that an adversary may intentionally send a preliminary group request to a member in order to monitor the member's profile update regularly. To avoid such a scenario, our approach includes an automated system by which a group member can set a time window while accepting a group formation request. If a user who does not send any query request to a group member by the time window after the group is formed, the group member leaves the group. A member can also leave a group before the time window if the group member knows any suspicious activity by the user. On the other hand, if a query is requested by the user, the time window can be extended. The trustworthiness and reputation issues for selecting group members and accepting group formation request of a user are discussed in detail in Section 6. In addition, when a query requestor forms G from F, the query requestor does not select all members of F due to the fear of loosing privacy. The query requestor selects those members from F with whom the query requestor feels comfortable to share her location.

4.2. Query Processing

In our crowdsourced approach, a user forms a group with similar-minded people and processes queries with the co-operation of the selected group members. We propose that the query evaluation process can be either parallel or sequential. According to *parallel processing*, a query is sent to the group members, and all the group members process the requested query on their local databases and return relevant POI and space information to the query requestor. The query requestor finds the query answer based on the received POIs from group members. The query requestor also computes the accuracy of the query answer in terms of the confidence level using the received space knowledge from the group members. On the other hand, in the *sequential processing*, a query is sent to a group member and the answer including the POI and space information is passed from one member to another. We will show that in the sequential

processing, a member can prune the POIs and space information that cannot be parts of the final answer before sending the POI and space information to another member. Thus, at the end, in case of the sequential processing, the query requestor receives the final POIs with the relevant space information and computes the confidence level for the answer.

In Section 4.2.1, we present the details of query processing in the database of a member's local device when a kNN query or a range query request is issued by a query requestor. We call these *search* procedures combinedly as a *single user processing*. In Section 4.2.2, we elaborate algorithms that a query requestor executes to determine the answer for a kNN or range query with the accuracy guarantee in the incomplete databases of group members.

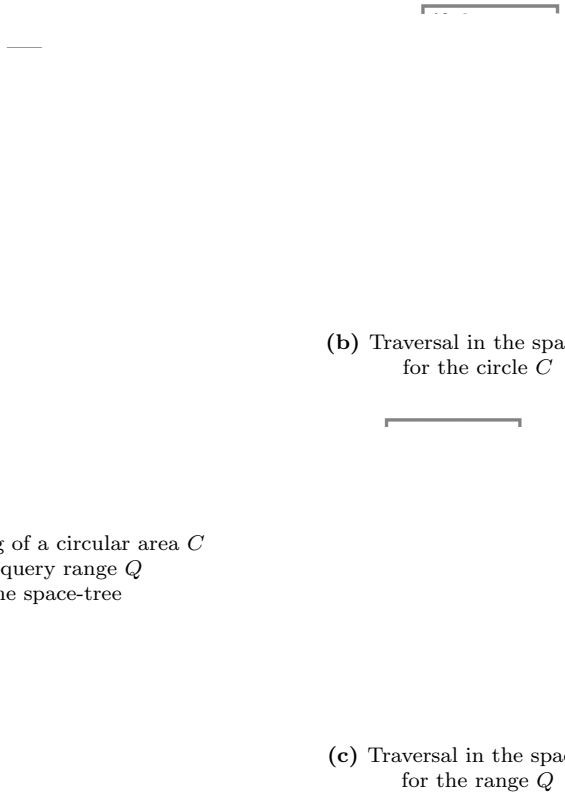
4.2.1. Single User Processing

A query requestor sends the group members a query location q and k for a kNN query, or a query range Q for a range query. As discussed in Section 3, the POI and space information are indexed using a POI-tree and a space-tree on the local devices of group members. After receiving a query request on the device of a group member, our approach executes a kNN or a range query processing algorithm in the POI-tree to find the query answers based on the available POI data on the user's device, and then retrieves the space knowledge relevant to the query as a set of minimum bounded rectangles (MBRs) from the space-tree.

For a kNN query, we use a best-first search (BFS) [2, 13] in the POI-tree to find the k POIs that have k smallest distances with respect to the query requestor's location q. After retrieving the POIs $P_i = \{p_1, p_2, \ldots, p_k\}$ from the local database of a member g_i, our approach determines the circular area C (similar area as shown in Figure 5(a)) with the centre at q and the radius equal to the distance between q and the k^{th} POI p_k in P_i. Next to find the known region within C, our approach executes a search in the space-tree. The search starts from the root of the space-tree and adds an MBR in the MBR set R_i if the MBR represented by a space-tree node has the flag $known = 1$, and the area of the MBR included within C is at least equal to a threshold limit, which we assume 85% in our experiments. The search continues to the child nodes if the MBR represented by the parent node has the flag $known = 0$, and the MBR overlaps with C. If an MBR represented by a space-tree node falls outside C, the search does not continue to its child nodes. Figure 5(b) shows the visited nodes of the space-tree for processing a query to find the known region within C. The MBRs R_{131}, r_{121}, r_{122} and R_{142} have flag $known = 1$, and are included in R_i. We name this procedure of evaluating a kNN query for a group member as *kNN_Search* and the query requestor calls this procedure as shown in Algorithm 1 in Section 4.2.2.

For a range query, we use the algorithm proposed in [4] to retrieve the set of POIs P_i from the POI-tree that fall within the query range Q (similar to Figure 5(a)). Then, our approach determines a set of MBRs from the space-tree such that an MBR in R_i has the flag $known = 1$, and the area of the MBR included within Q is at least equal to a threshold limit, which we assume 85% in our experiments. To determine R_i, our approach uses the same search procedure

in the space-tree as described earlier for a kNN query. Figure 5(c) shows the visited nodes of the space-tree for processing a query to find the known region within Q. The search in the space-tree returns MBRs R_{313}, r_{311} and r_{314} as the set R_i. We refer this procedure as *Range_Search* and the query requestor calls this procedure as shown in Algorithm 2 in Section 4.2.2.



(a) Mapping of a circular area C and a query range Q in the space-tree

(b) Traversal in the space-tree for the circle C

(c) Traversal in the space-tree for the range Q

Figure 5: Query processing in the space-tree

In our proposed system, for the parallel processing each group member g_i returns $\{P_i,\ R_i\}$ independently to the query requestor. Thus, there can be overlap among the received information from group members. For a kNN query the query requestor has to find the k nearest POIs from the received POIs. For a range query, the query requestor has to ensure that no POI is included more than once in the query answer. After finding the final query answer, the query requestor computes the confidence level based on the received space knowledge to determine the accuracy level of the query answer. In case of the sequential processing, the query requestor does not have to do these steps except the computation of the confidence level.

For the sequential processing, a group member can receive the query from the query requestor or another group member. If a group member g_i receives directly from the query requestor, then g_i computes $\{P_i,\ R_i\}$, marks the visited

flag associated with her ID as true, and forwards $\{P_i, R_i\}$ along with the query (q and k for a kNN query or Q for a range query) to another member in the group whose visited flag is false. Here, initially each POI of P_i is associated with the ID of g_i and will be updated if another group member posses the POI. If g_i receives $\{P_j, R_j\}$ from another member g_j in the group, g_i determines $\{P_i, R_i\}$, and then updates $\{P_i, R_i\}$ by merging it with $\{P_j, R_j\}$. The member g_i ensures that no POI or space knowledge is included more than once and prunes the POIs and MBRs that cannot be parts of the query answer. If any POI is in both of P_i and P_j, then the ID of g_j is associated with the POI. The member g_i includes any MBR r in R_i, if the percentage of the area of r, covered by the relevant range (C or Q), is at least equal to a threshold limit. In addition, g_i checks the possibility to combine small MBRs to form a single large MBR as the communication overhead increases with the increase of the number of MBRs.

The POI-set P_i is updated as $P_i \cup \{P_j - P_i\}$. For a kNN query, k POIs in P_i that have k smallest distances from q are kept in P_i and other POIs are removed from P_i. If the k^{th} nearest POI comes from P_j, the radius of the already computed circular area C is updated to the k^{th} smallest distance. If C becomes smaller, any MBR in R_i or R_j whose area covered by C becomes smaller than the threshold limit is removed from R_i or R_j, respectively.

For each MBR r in R_j, our approach checks whether it is covered by any MBR in R_i. If yes, the MBR r is not included in R_i. Otherwise, our approach checks whether r includes any MBR in R_i. If yes, the MBR in R_i that is covered by r is removed from R_i and r is added to R_i. If not, r is added to R_i. After updating R_i, g_i checks whether it is possible to decrease the number of MBRs by merging them. For example, if there are two MBRs r_1 and r_2 with coordinates $< x_1, y_1, x_2, y_2 >$ as $< 12, 34, 60, 90 >$ and $< 45, 34, 304, 90 >$, respectively, they are combined into a single MBR with coordinates $< 12, 34, 304, 90 >$.

After updating $\{P_i, R_i\}$, our approach marks the visited flag associated with the ID of member g_i as true, and forwards $\{P_i, R_i\}$ along with the query to another member in the group whose visited flag is false. In case if a member does not find any other member in the group with the visited flag false, the member forwards $\{P_i, R_i\}$ to the query requestor.

(a) Received $\{P_j, R_j\}$ along with the query from a group member g_j

(b) Computed $\{P_i, R_i\}$ on local device of a group member g_i

(c) Updated $\{P_i, R_i\}$ by merging it with $\{P_j, R_j\}$ and then filtering

Figure 6: Computation of $\{P_i, R_i\}$ for a kNN query ($k = 2$)

Figure 6 shows an example for sequential processing of a kNN query. Let a group member g_i receives a kNN query ($k = 2$) with $\{P_j, R_j\}$ from another group member g_j, where $P_j = \{(p_{j1}, g_j), (p_{j2}, g_j)\}$ and $R_j = \{r'_{j1}, r'_{j2}, r'_{j3}\}$ (Figure 6(a)). After receiving the kNN query, g_i computes the POI-set $P_i = \{(p_{i1}, g_i), (p_{i2}, g_i)\}$ and the relevant MBR-set $R_i = \{r'_{i1}, r'_{i2}, r'_{i3}\}$ as shown in Figure 6(b). After merging P_i with received P_j, p_{j1} becomes the 2^{nd} closest POI from q. Since p_{i2} and p_{j2} have greater distance than the 2^{nd} nearest POI p_{j1}, according to the update process, (p_{i2}, g_i) and (p_{j2}, g_j) are discarded and C is updated with the radius $dist(q, p_{j1})$. Thus the updated POI-set becomes $P_i = \{(p_{i1}, g_i), (p_{j1}, g_j)\}$. Similarly, g_i determines the relevant MBR-set R_i by merging it with R_j and the MBR-set R_i becomes $\{r'_{i1}, r'_{i2}, r'_{i3}, r'_{j1}, r'_{j2}, r'_{j3}\}$. From this MBR-set, g_i finds that r'_{i1} and r'_{j1} fall outside C and it is possible to merge r'_{i2} and r'_{j2} into a single MBR r_{i2}. Therefore, the updated MBR-set becomes $R_i = \{r_{i1}, r_{i2}, r_{i3}\}$, where r'_{i3} and r'_{j3} are renamed as r_{i1} and r_{i3}, respectively. Then g_i sends $\{P_i, R_i\}$ to the query requestor or the next member whose visited flag is false. Figure 6(c) shows the updated $\{P_i, R_i\}$.

4.2.2. Algorithms

In this section, we present the algorithm that a query requestor executes to process a query in cooperation with her group members. Algorithm 1 shows the pseudocode to process a kNN query with the parallel processing. The inputs to the algorithm are a query location q and k, and the output is the query answer P that includes k POIs that have k smallest distances with respect to q and the confidence levels M_1 and M_2. The algorithm starts with initializing a set of MBRs S, and a set of POIs P to ϕ. Then the query requestor sends the query to the selected group members using Function *kNN_Search* in Line 3. Each member g_i returns a set P_i of k nearest POIs from q based on the POI knowledge of g_i, and relevant space knowledge as a set R_i of known MBRs. It may happen that the query requestor receives same POIs or MBRs from more than one group members. Thus, the query requestor has to aggregate the POI and space knowledge by ensuring that no POI or MBR is included more than once in the merged sets. The query requestor also needs to find k nearest POIs with respect to q from the received POI sets $P_1, P_2, \ldots, P_n$, and prune POIs and MBRs that are not relevant to the final query answer. To perform the above mentioned tasks, Function *Aggregate* is called in Line 5 of Algorithm 1. Function *Aggregate* returns the query answer, i.e., P and the space knowledge S for computing the confidence levels of the query answer. Finally, the query requestor computes confidence levels for the query answer using Function *Compute_Confidence_NNQ* (Line 6). The procedure to compute confidence levels for a kNN query is discussed in Section 4.2.3.

Algorithm 2 shows the pseudocode to process a range query with the parallel processing. The input to the algorithm is a query range Q, and the output is the query answer P that includes POIs whose locations fall in Q and the confidence level M_1. Algorithm 2 works in a similar way to Algorithm 1. Algorithm 2 uses Function *Range_Search* in Line 3 to send the query request to the selected group members. On the other hand, Function *Aggregate* only needs to ensure

Algorithm 1 :Parallel_Process_kNN_Query

Input : q, k
Output : Answer-set P, Confidence levels M_1 and M_2
1: $S \leftarrow \phi$, $P \leftarrow \phi$
2: **for** each $g_i \in G$ **do**
3: $\quad \{P_i, R_i\} \leftarrow kNN\_Search(k, q)$
4: **end for**
5: $\{P, S\} \leftarrow Aggregate(q, P_1, P_2, \ldots, P_n, R_1, R_2, \ldots, R_n)$
6: $M_1, M_2 \leftarrow Compute\_Confidence\_NNQ(q, P, S)$
7: **return** P, M_1, M_2

that no POI or MBR is included more than once in the merged sets P and S. Note that for a range query, all POIs returned by the group members are the part of the query answer as they fall inside the range Q. Finally, the query requestor computes confidence level for the query answer using Function *Compute_Confidence_RQ* (Line 6). The procedure to compute confidence level for a range query is discussed in Section 4.2.3.

Algorithm 2 :Parallel_Process_Range_Query

Input : Q
Output : Answer-set P, Confidence level M_1
1: $S \leftarrow \phi$, $P \leftarrow \phi$
2: **for** each $g_i \in G$ **do**
3: $\quad \{P_i, R_i\} \leftarrow Range\_Search(Q)$
4: **end for**
5: $\{P, S\} \leftarrow Aggregate(Q, P_1, P_2, \ldots, P_n, R_1, R_2, \ldots, R_n)$
6: $M_1 \leftarrow Compute\_Confidence\_RQ(Q, S)$
7: **return** P, M_1

For the sequential processing of a kNN or a range query, algorithms are similar to those of parallel processing except that the query requestor sends the IDs of group members and a visited flag initialized as false for every group member along with the query to a single group member. The query requestor receives the final query answer and relevant space knowledge from a group member instead of all. Thus, there is no aggregation process to ensure that no POI or MBR is included more than once or to find the final POIs. After receiving a POI-set and an MBR-set from a group member, the query requestor directly computes the confidence levels for the query answer.

4.2.3. Confidence Level

For a kNN query answer, our approach determines two types of confidence levels to assess the quality of the query answer: *area based* and *distance based confidence levels*. For a range query, our approach only considers the area based confidence level to assess the quality of the answer as no distance computation related to POIs are involved in the range query. In case of a kNN query, we measure the confidence levels for each of the POIs included in the query answer and then take the average of these k confidence levels as the confidence level for the query answer.

Let p_i be the i^{th} nearest POI from q included in the answer-set P, and C_i represents the circular area at center q with the radius equal to the distance between q and p_i, $dist(q, p_i)$. S_i represents the set of MBRs (i.e., known spaces) that intersect with the circular area C_i. $Area(C_i)$ denotes the area of C_i and $Area(S_i)$ denotes the total area of the MBRs that reside within C_i. The area based confidence level for a POI p_i is defined as follows:

$$M_{1i} = \frac{Area(S_i)}{Area(C_i)}, \tag{1}$$

The area based confidence level for the kNN query answer is computed as $M_1 = \sum_{i=1}^{n} M_{1i}/k$. For a range query, S denotes the set of known MBRs in the query range Q, and $Area(Q)$ denotes the area of Q. $Area(S)$ represents the total area of the MBRs that reside within Q. The area based confidence level for a range query is defined as follows:

$$M_1 = \frac{Area(S)}{Area(Q)}, \tag{2}$$

(a) Measure for a kNN query **(b)** Measure for a range query

Figure 7: Computation of confidence level

On the other hand, the distance based confidence level for a kNN query ensures that there is no POI within a certain distance limit. If the distance based confidence level for the i^{th} nearest POI is 0.5 then it ensures that there is no other POI that has distance from q less than the half of the distance between q and p_i. Assume that q is located in the known space, i.e., an MBR in S_i. Let d_1 represents the distance of q from the closest boundary of the known space as shown in Figure 7(a) and d_2 represents the distance between q and p_i, $dist(q, p_i)$. The distance based confidence level for a POI p_i is defined as follows.

$$M_{2i} = \frac{d_1}{d_2}, \tag{3}$$

The distance based confidence level for any POI is 0 if the known space does not include the query location q. Again, the distance based confidence level for the kNN query answer is computed as $M_2 = \sum_{i=1}^{n} M_{2i}/k$. The distance d_1 remains

constant for any POI included in the answer-set whereas d_2 increases with the increase of the distance between q and a POI. Thus, the confidence level for the nearest POI has the highest confidence level, and the k^{th} nearest POI has the lowest confidence level. Figure 7 shows examples of computing confidence levels for a kNN query and a range query. The shaded MBRs within the respective ranges are known and contribute in the confidence level computation.

5. Experiments

We run extensive experiments using a real dataset in a simulated environment to validate the effectiveness of our approach. We use 104770 POIs of 63 types of California [15] as real datasets. The data space is normalized into a span of 10,000 X 10,000 square units. We index POI and space knowledge of each user in memory using a R-tree and a quad-tree, respectively. Since there exists no approach in the literature to evaluate kNN and range queries in incomplete databases with crowdsourced data and computation, we perform a comparative analysis between parallel and sequential processing of our approach in terms of communication and computational overhead. The computational time represents the total processing time required for a query. To approximate the communication overhead irrespective of the bandwidth of the network used for the communication among the group members, we measure the size of the information exchanged among the group members in bytes. We also measure the confidence levels obtained for kNN and range queries by varying different parameters in our experiments.

In the simulated environment, we assume that groups are already formed. To simulate the real environment, in our experiments, group members do not have knowledge of all POIs. We introduce the parameter *POI knowledge* to measure the percentage of POI knowledge that group members together have. For example, for 60% POI knowledge, we randomly select 62862 POIs (60% of 104770 POIs) from California dataset and distribute them among considered number of group members in an experiment. Furthermore, in reality, more than one users can have knowledge of same POIs, i.e., group members may have overlapping POI knowledge. We introduce the parameter *overlap* to measure the percentage of overlap of POI knowledge among group members. For example, if there is 5% overlap in 60% POI knowledge, then we randomly select 3143 POIs (5%) from 62862 POIs (60% of 104770 POIs) and store each of these POIs in more than one group members' databases. In addition, group members may have the knowledge that they know all POIs of some areas in the total space, which is expressed using *space knowledge*. x% space knowledge represents that group members together know x% of the total space. In addition to POI knowledge, overlap, space knowledge, we also vary group size, k, and range in our experiments. We summarize these metrics with their meanings, the range and default values in Table 1.

We vary the group size from 2 to 64. Considering the fact that a query requestor forms groups only with trusted and similar minded people, it is a resealable assumption that the query requestor does not rely on a large number

of people to form the group. Therefore, we set the default group size to 8. We vary the POI knowledge(%) and space knowledge(%) up to 100% and 80%, respectively. It may be possible for a group to have combined knowledge of all POIs, but it may not be a realistic assumption that group members also know the fact that they know all POIs. Moreover, when we vary the space knowledge, we set the POI knowledge to its default value. For 100% space knowledge of a group, the POI knowledge of the group must be 100%. Since the default value of the POI knowledge is set to less than 100% to simulate incomplete databases, it is not possible to vary the space knowledge up to 100%. We run preliminary experiments to determine the maximum possible value for the space knowledge for the default POI knowledge, and we find that the maximum space knowledge that the group members together can have is slightly higher than 80%. Thus, we set the maximum space knowledge of group members to 80%. For overlap parameter, we vary it in a wide range from 5% to 80%. For kNN queries, we vary k from 3 to 8 and set the default value of k to the middle value 5. It is expected that the query requestor does nor request for a large value of k because the distance between the query requestor and the k^{th} POI increases for a larger value of k. On the other hand, for the range query we select the range in a realistic manner; for example, the default value of the range 0.005% of the total space represents the area of a small suburb in California.

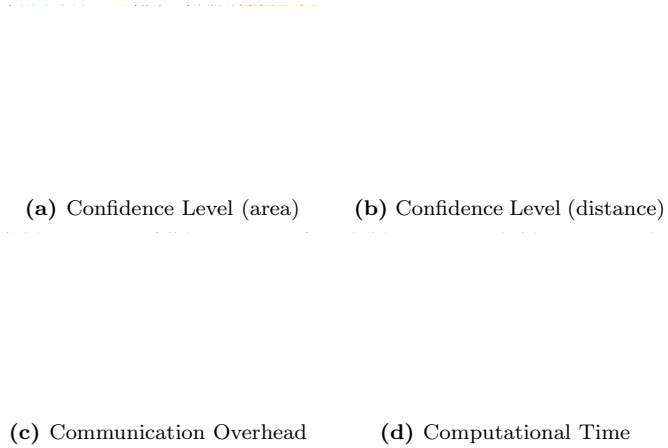
Metric Name	**Meaning**	**Range**	**Default Value**
Group Size	Number of selected group members for a query	2-64	8
POI Knowledge (%)	Percentage of known POI data	50%-100%	60%
Overlap (%)	Percentage of overlap of POI knowledge among group members	5%-80%	5%
Space Knowledge (%)	Percentage of known space data	30%-80%	70%
k	Number of NNs	3-8	5
Range (%)	Percentage of search space in the total space	0.001%-0.01%	0.005%

Table 1: Different metrics

We use Monte Carlo simulation to perform our experiments. For every experiment, we compute 100 sample queries, process them and measure the average performance in terms of confidence levels, communication overhead, and computational time. We implemented the application in java, and run all experiments using a computer with Intel Core i5 2.30 GHz CPU and 4GB RAM. In Sections 5.1–5.6, we present our experimental results for varying group size, POI knowledge, space knowledge, overlap, k, and range respectively. In Section 5.7, we compare the crowdsourced approach with the traditional systems where an LSP is responsible to evaluate location-based queries.

5.1. Effect of Group Size

In this set of experiments, we vary the number of group members *Group Size* n as 2, 4, 8, 16, 32, 64 for both parallel and sequential processing of kNN and range queries. Figure 8 shows the performance of our approach for varying the group size.



(a) Confidence Level (area) **(b)** Confidence Level (distance)

(c) Communication Overhead **(d)** Computational Time

Figure 8: Effect of Group Size

Figure 8(a) shows the confidence level obtained for both kNN and range queries with respect to the area measure, and Figure 8(b) shows the confidence level obtained for kNN queries with respect to the distance measure. Both confidence level measures remain constant with the increase of n for both types of queries. This is because we set POI knowledge and space knowledge to default values while varying the group size. Thus, if other parameters remain constant, the confidence level does not depend on the increasing group size.

Figure 8(c) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 8(d) shows that the computational time for sequential processing is higher than that for parallel processing, which are expected. We know that in case of the parallel processing, there can be overlap of POI and space knowledge received from the group members, which causes increase in the communication overhead. In case of the sequential processing, group members participate one after another, perform pruning and merging in the POI and MBR sets, and thereby increase the overall processing time. On the other hand, both the communication overhead and the computational time increase with the increase of the group size. This is because the amount of overlapped data shared among group members increases with the increase of the number of group members.

5.2. Effect of POI Knowledge

In this set of experiments, we vary the percentage of known POI data, i.e., *POI Knowledge(%)*, as 50%-100% for both parallel and sequential processing of

(a) Confidence Level (area) **(b)** Confidence Level (distance)

(c) Communication Overhead **(d)** Computational Time

Figure 9: Effect of POI Knowledge (%)

kNN and range queries. Figure 9 shows the performance of our approach for varying the POI knowledge.

Both graphs in Figure 9(a) and (b) show that the confidence level increases with the increase of the POI knowledge, which is expected. However, we observe that the confidence level is not 100% when the POI knowledge is 100%. This is because of the incomplete space knowledge of group members (default space knowledge is 70%), i.e., group members are not aware of the fact that they together know all POIs of the total space. Therefore, in spite of having 100% POI knowledge, the confidence level may not increase up to 100%.

Figure 9(c) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 9(d) shows that the computational time for sequential processing is higher than that for parallel processing. On the other hand, both the communication overhead and the computational time increase with the increase of the completeness of databases because of the increased number of POIs.

5.3. Effect of Space Knowledge

In this set of experiments, we vary the percentage of known space data, i.e., *Space Knowledge(%)*, as 30%-80% for both parallel and sequential processing of kNN and range queries. Figure 10 shows the performance of our approach for varying the space knowledge. We set the maximum space knowledge as 80% because in our experiments we found that the maximum space knowledge that group members can have for the default 60% POI knowledge is slightly higher than 80%. Figures 10(a) and (b) show that the confidence level increases with the increase of the space knowledge.

Figure 10(c) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 10(d) shows that the computational time for sequential processing is higher than that for parallel processing. The more the space knowledge is, the more processing time is required to execute queries and the more is the number of MBRs. Thus with

(a) Confidence Level (area) **(b)** Confidence Level (distance)

(c) Communication Overhead **(d)** Computational Time

Figure 10: Effect of Space Knowledge (%)

the increase of space knowledge, both of communication overhead and computational time increase. The increase of the communication overhead is not significant because nearby small MBRs are merged to a single MBR.

5.4. Effect of Overlap

In this set of experiments, we vary the percentage of overlap of POI knowledge among group members as 5%, 10%, 20%, 40%, 80% for both parallel and sequential processing of kNN and range queries. Figure 11 presents the comparative graphs for these queries.

(a) Confidence Level (area) **(b)** Confidence Level (distance)

(c) Communication Overhead **(d)** Computational Time

Figure 11: Effect of Overlapped data among group members

Figure 11(a) presents the confidence level obtained for both kNN and range queries with respect to the area measure and Figure 11(b) presents the confidence level obtained for kNN queries with respect to the distance measure. We observe that both the confidence levels remain constant with the increase in

the overlapped percentage of databases. This is because POI and space knowledge are set to default values while varying the overlapped percentage of POI knowledge among group members.

Similar to previous experiments, Figure 11(c) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 11(d) shows that the computational time for sequential processing is higher than that for parallel processing. With the increase of overlapping percentage, both of the communication overhead and the computational time increase for both kNN and range queries.

5.5. Effect of k

In this set of experiments, we vary the value of k from 3 to 8 and execute both parallel and sequential processing of kNN queries. The metric k has no effect for Range queries. The graphs in Figure 12 present the effects of k in kNN queries.

(a) Confidence Level (area) **(b)** Confidence Level (distance)

(c) Communication Overhead **(d)** Computational Time

Figure 12: Effect of k

Figure 12(a) shows the confidence level obtained for kNN queries with respect to the area measure, and Figure 12(b) shows the confidence level obtained for kNN queries with respect to the distance measure. Both confidence levels vary almost linearly with the increase of k. With the increase of k, the area based confidence level linearly increases because the search area for the known space increases and for a larger search area, the area of the know space also increases. On the other hand, the distance based confidence level decreases with the increase of k, because the distance of the closest boundary of the known space from the query location, i.e., d_1 in Equation 3, remains constant with the increase of k.

Similar to other experiments, Figure 12(c) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 12(d) shows that the computational time for sequential processing is higher than that for parallel processing. In Figure 12(c), we observe that with

the increase of k, the communication overhead slowly increases for both parallel and sequential processing of kNN queries.

5.6. Effect of Range

In this set of experiments, we vary *Range (%)* as 0.001%-0.01% for both parallel and sequential processing of range queries. Since the range is not specified for a kNN query, we only show the effect of varying range for range queries in Figure 13.

(a) Confidence Level (area) **(b)** Communication Overhead **(c)** Computational Time

Figure 13: Effect of Range (%)

In Figure 13(a), we observe that the confidence level increases with the increase of the range. This is because with the increase of the range, the probability of the intersection area of the known space with the range also increases. Figure 13(b) shows that the communication overhead is less for sequential processing than that for parallel processing and Figure 13(c) shows that the computational time for sequential processing is higher than that for parallel processing, which are expected. Both communication overhead and the computational time for range queries rapidly increase with the increase of the range because of expanding the search area in the space-tree.

5.7. Comparison with the traditional LSP-based systems

We propose a novel approach that eliminates the role of LSPs for evaluating LBSs, and protects privacy of users from untrusted LSPs. Both data and computations are crowdsourced in our system. In the traditional systems to evaluate LBSs, it is assumed that LSPs know all POIs of an area, and the queries are evaluated based on the LSP's POI knowledge. In the proposed system, we assume that a group of crowd who evaluate a query may not know all POIs. Thus, a location based query may need to be evaluated in incomplete databases. To address data incompleteness, our system computes the confidence level of the query answer (see Section 4.2.3), which may vary depending on the POI and space knowledge of group members. In the experiments, we evaluate the confidence level of the query answers for different settings of parameters. For the traditional LSP-based systems, the confidence level of the query answer would be 100%, if the assumption of knowing all POIs by an LSP holds in the reality. Our approach protects privacy of users with respect to the LSPs, which in turn results in the slight decrease of the confidence level of the query answers.

Furthermore, an untrusted LSP can be biased for some POIs, and returns incorrect query answers. In our crowdsourced approach, groups are formed with

trusted and similar minded people, and queries are evaluated independently by group members. Thus, the probability is very low that every group member intentionally returns wrong POIs to the query requestor, and the final answer from the aggregated data is biased. In addition, in the proposed approach, since a query requestor forms groups with the similar minded people, query answers implicitly ensure the query requestor's choice and preference for POIs.

6. Trustworthiness and Reputation

Ensuring the trustworthiness and reputation of the group members is an important factor in the crowdsource based system, which provides a user the confidence to rely on others for the required service. Low reputable or trustworthy users can provide low quality or malicious answers. We propose an idea to develop a rating system that measures trustworthiness and reputation in terms of ratings. Users of the crowdsourced system can use the ratings while sending group formation request and accepting group formation request.

Each user has two ratings: global and local. The global rating is measured based on the aggregated feedback of query requestors who interact with the user. The local rating of a user is done by a query requestor. A user can have multiple local ratings if the user serves more than one query requestors. A query requestor can rate a user by using a Likert scale and select one among five options: very bad (1), bad (2), neutral (3), good (4), and very good (5). Initially, the local and global ratings of a user can be set to neutral (3). A query requestor may rate a user when the user serves as a member of G for the query requestor. Thus, a query requestor can rate a same user multiple times. The local rating with respect to a query requestor and the global rating of a user are updated when a query requestor rates a user. The local rating by a query requestor is computed as an average of all ratings provided by the query requestor. If a user serves m query requestors and has m local ratings, the global rating of the user is computed as $\sum_{i=1}^{m} lr_i/m$.

The rating of a user is done by a query requestor based on the retrieved answer from the user during a query evaluation process. Usually, a query requestor visits one POI among received POIs as the query answer. If the visited POI meets the satisfaction level of the query requestor, the query requestor can rate the group member (or group members) who provided the POI as very good (5). A query requestor may rate a group member as neutral (3), if the final query answer does not include any POI of the group member. In case of parallel processing a query requestor independently receives POIs from every group member. Thus, the query requestor knows about which POI is sent by whom. In case of sequential processing, since the query requestor does not directly receives POIs from the group members, and thus, it is not possible for the query requestor to know who has sent which POIs. To facilitate the rating system, we incorporate the IDs of group members with every POI, who send the POI as a part of the query answer based on their local databases.

The rating system can be specialized based on other factors like location and POI types (e.g., restaurants, shopping centers, or hospitals). It may happen

that a user is knowledgable of POIs in the areas where she lives or works, and does not know POIs in other areas. Similarly, a user can know more about restaurants than hospitals. Thus, if the ratings are computed based on area or POI type, then it will have more values.

Query requestors of our proposed crowdsourced system can use local and global ratings while forming groups. It may happen that a user u_i, who wants to form a group, does not know another user u_j personally and never rated u_j. In this scenario, in addition to the global rating, u_i may compute a weighted rating, and consider it to decide whether a group formation request would be send to u_j. Both global and weighted ratings are computed using local ratings of users. A user may have multiple local ratings assigned by different query requestors. The intuition behind computing the weighted rating is to give different weights to these local ratings. Thus, the weighted rating may vary for a user depending on the weights assigned for local ratings. Suppose u_i computes the weighted rating for u_j. Let lr_{kj} represents a local rating of u_j given by a query requestor u_k. Then the weight for lr_{kj} is lr_{ik}, where lr_{ik} represents the local rating of u_k given by u_i. Thus, u_i assigns more weight to a local rating of u_j, if u_i finds that the source of the local rating is more trustable to her.

Let U_{ij} be the set of familiar users to u_i with whom u_j is also familiar, u_i rates every user $u_k \in U_{ij}$ as lr_{ik}, and u_k rates u_j as lr_{kj}. The weighted rating wr_{ij} for u_j given by u_i is computed as follows:

$$wr_{ij} = \frac{\sum_{u_k \in U_{ij}} (lr_{ik} * lr_{kj})}{\sum_{u_k \in U_{ij}} lr_{ik}}, \tag{4}$$

In the similar way, a user of the system can decide whether the user should accept a group formation request based on local, global and weighted ratings. Specifically, the weighted rating helps a user to avoid collusive attacks while accepting a group formation request. For example, a group of users may collude because of mutual interests or monetary incentives, rate themselves to improve their local and global ratings, and thereby mislead people. A weighted rating gives higher priorities to those local ratings that are given by trusted users.

As mentioned earlier, the local ratings are given by the query requestors are based on the answers received from the group members. Thus, we have to ensure that the data that a group member sends to a query requestor is not modified by others. Next we discuss a technique to authenticate the data that a group member provides is received by the query requestor in an intact form.

6.1. Authentication

In case of parallel processing, the query requestor directly receives the answer from every group member in G, and thus, there is no scope of modification of query-answers by other members. In the sequential processing, the query is processed sequentially by group members. To remind the reader, the query requestor sends the query to a group member in G, who processes the query, and forwards the query with the answer to another member. A member who received the query and the answer, determines an answer of the query on her

database, merges the answer with the received one, and updates the query answer if it can be improved. Then the member forwards the query with the answer to another group member. The process continues until all group members have evaluated the query on their local databases. The details process is described in Section 4.2.1. Thus in the sequential processing, a group member can intentionally manipulate the query answer received from other members. Such manipulation or fraud detection requires an authentication process.

A group member g_i can manipulate the answer received from another group member g_j in three ways: (i) associate the ID of g_j with a POI that has not been sent by g_j, (i) remove the ID of g_j for a POI that has been sent by g_j, and (iii) remove both POI and associated ID of g_j, which should be included as a part of the query answer. To authenticate that a group member has not intentionally modified the message, the following technique can be incorporated in our crowdsourced approach.

Figure 14: Query authentication process

After the final query answer is returned to the query requestor, the query requestor may randomly select a group member to verify. Without loss of generality, let the query requestor randomly select a group member g_3. For the authentication purpose, the query requestor sends a message to the group member g_2 who forwarded the query answer to g_3, and the group member g_4 who received the query answer from g_3 as shown in Figure 14. In response to the message g_2 sends the query answer that g_2 forwarded g_3, and g_4 sends the query answer that g_4 received from g_3. Then, the query requestor compares the answers of g_2 with g_4, and can verify whether any manipulation is done by g_3.

For example, in response to a range query, if g_2 sends g_3 the POI-set $\{(p_2, g_1, g_2), (p_7, g_1)\}$, and g_4 receives from g_3 the POI-set $\{(p_2, g_1, g_2), (p_7, g_1, g_3), (p_8, g_3)\}$, then no manipulation has been done by g_3. However, if g_4 receives from g_3 either $\{(p_2, g_1), (p_7, g_1, g_3), (p_8, g_3)\}$ or $\{(p_7, g_1, g_3), (p_8, g_3)\}$, then g_3 is detected as a malicious member.

In another scenario, in response to a kNN ($k = 2$) query, if g_2 sends g_3 the POI-set $\{(p_2, g_1, g_2), (p_7, g_1)\}$ and g_4 receives from g_3 the POI-set $\{(p_2, g_1, g_2), (p_8, g_3)\}$, then no manipulation has been done by g_3, if $dist(q, p_2) < dist(q, p_8) < dist(q, p_7)$. If g_4 receives from g_3 either $\{(p_2, g_1), (p_8, g_3)\}$ or $\{(p_7, g_1, g_3), (p_8, g_3)\}$, then the member g_3 is detected as a malicious member.

To penalize a malicious member, the ratings (both local and global) of the

member can be reduced to the lowest value (i.e., 1). Because of low ratings of a user, there is a high probability that no one will send or accept group formation requests from that malicious user. Since a group member for the authentication purpose is selected randomly, all group members will fear to do any intentional manipulation because of losing ratings. However, if more than one group members (e.g., g_2, g_3 and g_4) together intentionally manipulate the query answer of another member (say g_1), then the discussed authentication technique fails, and we need to involve more than two users in the authentication process. On the other hand, the communication overhead of the sequential process increases with the increase of the number of group members in the authentication process. Since the reduced communication overhead is the only advantage of the sequential processing over the parallel processing technique, in such a scenario the parallel processing becomes preferable due to its reduced processing cost of the query answer.

7. Related Work

kNN queries [2, 3] and range queries [4, 5] have been extensively addressed in the literature. In [2] and [16], the authors proposed *Depth First Search* (DFS) and *Best First Search* (BFS) using the R-tree for efficient processing of kNN queries, respectively. For unknown values of k, BFS can be used to incrementally evaluate the next set of NNs. In [3], Yu et al. proposed an efficient approach to evaluate kNN queries in the high-dimensional space. Kolahdouzan et al. [17] proposed a novel approach to efficiently evaluate kNN queries in spatial network databases by using a first order Voronoi diagram [18]. In [19, 20, 21], the authors developed approaches to evaluate kNN queries with respect to inaccurate or imprecise locations of users.

In [5], Pagel et al. investigated the performance of range queries using different data structures. In [4], Tao et al. developed an approach for processing range queries on uncertain data. In [22], the authors proposed an approach to find POIs that are located within the specific range and the textual description of POIs match with the specified spatial keywords. However, all of these approaches for kNN and range queries assume that the POI data is stored in the LSP's database and the query is evaluated by the LSP. Users do not feel comfortable to share their location data in precise or imprecise form with untrusted LSPs. Our approach for evaluating LBSs is based on crowdsourced data and computation without involving an LSP. We assume that a location-based query is processed in distributed and incomplete databases of the crowd.

A very closely related crowdsource based work to ours is [12]. In this work, the authors developed a crowdsource based approach for group nearest neighbor queries that finds the POI that has the minimum aggregate distance from the group members. However, the main focus of [12] is to preserve location privacy of users from other group members and does not consider the incompleteness of POI data. In [23], the authors proposed an approach that caches the queries and answers retrieved from the LSP, evaluates kNN queries based on the stored data of users connected with the query requestor through wireless ad-hoc networks,

and checks whether it can be guaranteed that the evaluated answer is correct. The goal of this work is to reduce the query processing load on the LSP's database, and does not provide any efficient framework to process kNN queries and compute the confidence level of the query answers by eliminating the LSP. To the best of our knowledge, our work is the first to consider processing kNN and range queries using both crowdsourced data and computation in incomplete distributed databases with accuracy guarantee.

Due to the benefits of involving crowd in solving problems with efficiency and efficacy, crowdsourcing has been investigated in recent years for data collection [24, 25], query processing [26, 27] and recommendation systems [28, 29]. In [24, 25], the authors proposed techniques to extract crowdsourced datasets from social systems like bookmarking or tagging systems. In [30], the authors developed a technique to collect comprehensive tempo-spatial data by involving a limited number of vehicles, whereas the focus of the mobile crowdsensing framework [31] is to reduce the load of crowd for manually collecting data while preserving the accuracy of the urban data.

In [32], the authors exploited crowd-power to analyze data for query computation and to improve the correctness of answers. In [26], the authors engaged crowd for answering multiple-choice questions and exploited lightweight machine learning techniques to improve aggregated quality of crowdsourced answers. In [27], Kaplan et al. involved crowd to plan a trip such as a place to visit in a vacation. Keles et al. [33] proposed a crowdsourced model to synthesize the rankings of POIs with respect to the keywords specified by a user. Cowdsourced data have been used for variant purposes like evaluating the quality of public transport systems [34], constructing energy models for smartphones [35], profiling shops in indoor spaces [36], and building indoor localization systems [37]. In [38], the authors focused on quality of data instead of quantity, and deployed a new crowdsensing framework for wireless indoor localization systems.

In [39], the authors implemented a location-based search engine from the activities of crowd in social networks. Similar location based recommendation systems that use crowdsourced data have been also developed in [28, 29]. All of the above works have shown the necessity and successful application of crowdsourced data. We replace the role of an LSP with crowd for processing kNN and range queries with the accuracy guarantee, which has not been considered before. Our approach allows users to protect location privacy from the untrusted and biased LSPs, and enjoy services from similar minded and trusted peers.

8. Conclusion and Future Work

In this paper, we introduced a novel idea to efficiently evaluate kNN and range queries with crowdsourced data and computation. To preserve location privacy and provide trustworthy result, our approach does not involve an LSP to process the queries. In our system, users store both POI and space knowledge on their local databases and share them while evaluating kNN and range queries with their friends and relatives. We developed techniques to compute the confidence level of the computed query answers in incomplete distributed

databases. In our experiments, we showed that our approach is scalable in terms of group size and ensures high confidence level for the query answers.

We showed comparative analysis in terms of processing time and communication overhead between parallel and sequential processing of kNN and range queries. The processing time is less in the sequential processing of kNN and range queries and does not increase with the increase of the number of group members. For kNN and range queries, the parallel processing is on average 1.24 times and 1.71 times faster than the sequential processing, respectively. For kNN queries, the parallel processing incurs on average 1.73 times communication overhead than that of the sequential processing. For range queries, the parallel processing incurs on average 1.06 times communication overhead than that of the sequential processing.

In this paper, we assume that the location of a POI is precisely known. Sometimes, it may happen that users know the imprecise locations of POIs instead of exact locations (e.g., a user may know that there is a hospital in a specific suburb of a city). In the future, we aim to handle imprecise locations of POIs. In addition, we plan to extend our work for road network spaces, and consider other types of location-based queries such as group nearest neighbor (GNN) [40] and group trip planning (GTP) [41] queries.

References

[1] L. Calderoni, D. Maio, P. Palmieri, Location-aware mobile services for a smart city: Design, implementation and deployment, JTAER 7 (3).

[2] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: SIGMOD, 1995, pp. 71–79.

[3] C. Yu, B. C. Ooi, K. Tan, H. V. Jagadish, Indexing the distance: An efficient method to kNN processing, in: VLDB, 2001, pp. 421–430.

[4] Y. Tao, X. Xiao, R. Cheng, Range search on multidimensional uncertain data, in: TODS, Vol. 32, 2007.

[5] B.-U. Pagel, H.-W. Six, H. Toben, P. Widmayer, Towards an analysis of range query performance in spatial data structures, in: PODS, 1993, pp. 214–221.

[6] L. Hu, Y. Jing, W. Ku, C. Shahabi, Enforcing k nearest neighbor query integrity on road networks, in: SIGSPATIAL, 2012, pp. 422–425.

[7] I. Bilogrevic, M. Jadliwala, K. Kalkan, J. Hubaux, I. Aad, Privacy in mobile computing for location-sharing-based services, in: PETS, 2011, pp. 77–96.

[8] J. Freudiger, R. Shokri, J. Hubaux, Evaluating the privacy risk of location-based services, in: FC, 2011, pp. 31–46.

[9] Y. Jing, L. Hu, W.-S. Ku, C. Shahabi, Authentication of k nearest neighbor query on road networks, TKDE 26 (6) (2014) 1494–1506.

[10] L. Hu, W.-S. Ku, S. Bakiras, C. Shahabi, Spatial query integrity with voronoi neighbors, TKDE 25 (4) (2013) 863–876.

[11] T. Hashem, L. Kulik, R. Zhang, Countering overlapping rectangle privacy attack for moving kNN queries, Information Systems 38 (3) (2013) 430–453.

[12] T. Hashem, M. E. Ali, L. Kulik, E. Tanin, A. Quattrone, Protecting privacy for group nearest neighbor queries with crowdsourced data and computing, in: UbiComp, 2013, pp. 559–562.

[13] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: SIGMOD, 1984, pp. 47–57.

[14] H. Samet, The quadtree and related hierarchical data structures, CSUR 16 (2) (1984) 187–260.

[15] Real data: http://www.cs.fsu.edu/ lifeifei/tpq.html.
URL `http://www.cs.fsu.edu/~lifeifei/tpq.html`

[16] G. R. Hjaltason, H. Samet, Ranking in spatial databases, in: SSD, 1995, pp. 83–95.

[17] M. R. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: VLDB, 2004, pp. 840–851.

[18] F. Aurenhammer, Voronoi diagrams - A survey of a fundamental geometric data structure, CSUR 23 (3) (1991) 345–405.

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

[20] T. Hashem, L. Kulik, Safeguarding location privacy in wireless ad-hoc networks, in: Ubicomp, 2007, pp. 372–390.

[21] T. Hashem, L. Kulik, "Don't trust anyone": Privacy protection for location-based services, PMC 7 (2011) 44–59.

[22] W. Li, J. Guan, S. Zhou, Efficiently evaluating range-constrained spatial keyword query on road networks, in: DASFAA, 2014, pp. 283–295.

[23] W. Ku, R. Zimmermann, Nearest neighbor queries with peer-to-peer data sharing in mobile environments, PMC 4 (5) (2008) 775–788.

[24] C. Körner, M. Strohmaier, A call for social tagging datasets, SIGWEB (Winter) (2010) 2:1–2:6.

[25] B. Markines, F. Menczer, A scalable, collaborative similarity measure for social annotation systems, in: HT, 2009, pp. 347–348.

[26] B. I. Aydin, Y. S. Yilmaz, Y. Li, Q. Li, J. Gao, M. Demirbas, Crowdsourcing for multiple-choice question answering, in: AAAI, 2014, pp. 2946–2953.

[27] H. Kaplan, I. Lotosh, T. Milo, S. Novgorodov, Answering planning queries with the crowd, PVLDB 6 (9) (2013) 697–708.

[28] B. M. Sarwar, G. Karypis, J. A. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: WWW, 2001, pp. 285–295.

[29] D. Quercia, N. Lathia, F. Calabrese, G. D. Lorenzo, J. Crowcroft, Recommending social events from mobile phone location data, in: ICDM, 2010, pp. 971–976.

[30] Y. Liu, J. Niu, X. Liu, Comprehensive tempo-spatial data collection in crowd sensing using a heterogeneous sensing vehicle selection method, PUC 20 (3) (2016) 397–411.

[31] L. Xu, X. Hao, N. D. Lane, X. Liu, T. Moscibroda, More with less: lowering user burden in mobile crowdsourcing through compressive sensing, in: UbiComp, 2015, pp. 659–670.

[32] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, R. Xin, Crowddb: Answering queries with crowdsourcing, in: SIGMOD, 2011, pp. 61–72.

[33] I. Keles, S. Saltenis, C. S. Jensen, Synthesis of partial rankings of points of interest using crowdsourcing, in: GIR, 2015, pp. 15:1–15:10.

[34] S. Tan, X. Wang, G. Maier, W. Li, Riding quality evaluation through mobile crowd sensing, in: PerCom, 2016, pp. 1–6.

[35] E. Peltonen, E. Lagerspetz, P. Nurmi, S. Tarkoma, Where has my battery gone?: A novel crowdsourced solution for characterizing energy consumption, Pervasive Comput. 15 (1) (2016) 6–9.

[36] X. Guo, E. C. L. Chan, C. Liu, K. Wu, S. Liu, L. M. Ni, Shopprofiler: Profiling shops with crowdsourcing data, in: INFOCOM, 2014, pp. 1240–1248.

[37] C. Wu, Z. Yang, Y. Liu, Smartphones based crowdsourcing for indoor localization, MC 14 (2) (2015) 444–457.

[38] R. Kawajiri, M. Shimosaka, H. Kahima, Steered crowdsensing: incentive design towards quality-oriented place-centric crowdsensing, in: UbiComp, 2014, pp. 691–701.

[39] P. Shankar, Y. Huang, P. Castro, B. Nath, L. Iftode, Crowds replace experts: Building better location-based services using mobile social network interactions, in: PerCom, 2012, pp. 20–29.

[40] D. Papadias, Q. Shen, Y. Tao, K. Mouratidis, Group nearest neighbor queries, in: ICDE, 2004, pp. 301–312.

[41] T. Hashem, T. Hashem, M. E. Ali, L. Kulik, Group trip planning queries in spatial databases, in: SSTD, 2013, pp. 259–276.