Countering Overlapping Rectangle Privacy Attack for Moving kNN Queries

Tanzima Hashem, Lars Kulik, and Rui Zhang

Department of Computing and Information Systems
University of Melbourne, Victoria, 3052, Australia
Tel.: +61 3 8344 1350
Fax: +61 3 9348 1184
{*thashem,lars,rui*}*@csse.unimelb.edu.au*

Abstract

An important class of LBSs is supported by the moving *k* nearest neighbor (MkNN) query, which continuously returns the *k* nearest data objects for a moving user. For example, a tourist may want to observe the five nearest restaurants continuously while exploring a city so that she can drop in to one of them anytime. Using this kind of services requires the user to disclose her location continuously and therefore may cause privacy leaks derived from the user's locations. A common approach to protecting a user's location privacy is the use of imprecise locations (e.g., regions) instead of exact positions when requesting LBSs. However, simply updating a user's imprecise location to a location-based service provider (LSP) cannot ensure a user's privacy for an M*k*NN query: continuous disclosure of regions enable LSPs to refine more precise location of the user. We formulate this type of attack to a user's location privacy that arises from overlapping consecutive regions, and provide the first solution to counter this attack. Specifically, we develop algorithms which can process an M*k*NN query while protecting the user's privacy from the above attack. Extensive experiments validate the effectiveness of our privacy protection technique and the efficiency of our algorithm.

Keywords: Confidence level, Moving *k*NN queries, Overlapping rectangle attack, Location privacy.

1. Introduction

Location-based services (LBSs) are developing at an unprecedented pace: they started as web-based queries that did not take a user's actual location into account (e.g., Google maps), and can nowadays be accessed anywhere via a mobile device using the device's location (e.g., displaying nearby restaurants on a cell phone relative to its current location). While LBSs provide many conveniences, they also threaten our privacy. The advent of Apple's iPhone OS and Google's Android operating systems allow third parties to run their applications and become location-based service providers (LSPs) for the users of mobile devices. Thus, the LSPs are not only limited to large corporations such as Google or Microsoft, but include small developers, companies or the employers. Since an LSP knows the locations of its users, a user's continuous access of LBSs enables the LSP to produce a comprehensive profile of the user's movement with a high degree of spatial and temporal precision, possibly over an extended period of time. From this profile, the LSP may infer private information about users. For example, an organization may provide an LBS for its employees, which accidentally reveals that one employee is at a different company for a job interview, a fact an employee might wish to hide. The problem of protecting privacy is exacerbated if the available information from the access of LBSs is linked with other data sources; in the example, if the LSP collaborates with the company then more specific information about the user's interview could be obtained. The threat to privacy is becoming more urgent as positioning devices become more precise, and a lack of addressing privacy issues may significantly impair the proliferation of LBSs [1, 2].

An important class of LBSs is supported by the moving *k* nearest neighbor (MkNN) query [3, 4], which continuously returns the *k* nearest data objects for a moving user. For example, a tourist may want to observe the five nearest restaurants continuously while exploring a city so that she can drop in to a preferred one of the nearest five at any time. Alternatively, an M*k*NN query may run as a background application in a user's mobile device and when the user needs *k* nearest data objects for her current location, the user can have the query answers instantly. As an example, a woman may request an M*k*NN query for the nearest public place such as restaurant or pub while walking through a street at night so that the woman can quickly bring herself to safety in case of any suspicious situations. However, accessing an M*k*NN query requires continuous updates of user locations to the LSP, which puts the user's privacy at risk. The user's trajectory (i.e., the sequence of updated locations) is sensitive data and reveals private information. For example, if the user's trajectory intersects the region of a liver clinic, then the LSP might infer that the user has a liver disease and incorrectly associate a habit of high consumption of alcohol. Common approaches to combating privacy threats are either to prevent an LSP from inferring a user's identity or to not reveal any sensitive data such as locations to the LSP. In this paper, we consider the scenarios when the user's identity is revealed, say, to allow for personalized query answers [5, 6] (e.g., the LSP can return only those restaurants as M*k*NN answers which provide a higher discount for the user's credit card). In these scenarios, K-anonymity techniques [7, 8] that hide a user's identity do not apply, and we only have the option to protect the privacy of the user's locations.

A popular technique to hide a user's location from the LSP is to let the user send an imprecise location (typically a rectangle containing the user's location) instead of the exact location [5, 9, 10]. This technique is effective when the user is not moving. However, when the user moves and continuously sends rectangles as location updates, the LSP can still approximate the user's location if it takes into account the overlap of consecutive rectangles, which poses a threat to the user's *location privacy* and we call it the *overlapping rectangle (privacy) attack*. Although different privacy preserving approaches [5, 9, 11, 12, 13] have been developed for continuous queries, none of them address the overlapping rectangle attack. We define the problem of protecting a user's location privacy from the overlapping rectangle attack in the M*k*NN query as the *private moving kNN (PMkNN) query* and this paper provides the first approach to solve this problem. Our key idea to counter the overlapping rectangle attack is to change the fact that the user must be located in the overlap of consecutive rectangles. In other words, by our approach the user may not be located in the overlap of consecutive rectangles when she issues the service request and therefore LSPs will not be able to refine her location. We achieve this through two novel strategies: *slightly sacrificing the quality of service* (i.e., the accuracy of query answers) and *retrieving a few extra data objects* from the LSP.

Our approach allows users to specify the accuracy for their query answers (possibly done automatically by mobile devices using a generic privacy setting), which is motivated by the following observation. In many cases, users are willing to accept answers with a slightly lower accuracy if they can gain higher privacy protection in return. For example, a tourist looking for the closest restaurant may not mind driving to a restaurant that is not the actual closest one, if a slightly longer trip considerably enhances the tourist's privacy. In this context, "lower accuracy" of the answers means that the returned data objects are not necessarily the k nearest data objects but might be a subset of the $(k + x)$ nearest data objects, where x is a small integer. Our approach guarantees that the returned objects' distances to the query point are within a certain *ratio* of the actual k^{th} nearest neighbor's distance. We define a parameter, called *confidence level*, to characterize this ratio.

For every update of a user's imprecise location (a rectangle) in a PM*k*NN query, the LSP provides the user with a candidate answer set that includes the specified number of nearest data objects (i.e., k nearest data objects) with the specified confidence level for every possible point in the rectangle. The crux of this privacy protection strategy is to specify higher values for (i) the confidence level, and/or (ii) the number of nearest data objects, than actually required by the user and not to reveal the required confidence level and/or required number of nearest data objects to the LSP. Since the user's required confidence level and/or the required number of nearest data objects are lower than the specified ones, the candidate answer set must contain the required query answers for an additional part of the user's trajectory, which is unknown to the LSP. The availability of the query answers for an additional part of the user's trajectory provides the user with an option to send a rectangle that does not overlap with the previous one. Even if the user requests a rectangle that overlaps with the previous one, there is no guarantee that the user is in the overlap at the time of requesting the rectangle because the user has additional query answers for the area outside the previous rectangle. Based on this idea, we develop an algorithm to compute the user's consecutive rectangles that counters the overlapping rectangle attack and prevents the disclosure of the user's locations. Although our approach for privacy protection works if either the required confidence level or the required number of nearest data objects is hidden, hiding both provides higher level of privacy. On the other hand, if a user does not want sacrifice her accuracy of answers then the user can opt for only hiding the required number of nearest data object, which means our approach can also provide exact query answers while protecting the user's location privacy. In summary, we make the following contributions.

- We identify an attack to a user's location privacy that arises from overlapping consecutive regions in an MkNN query. We propose the first solution for PMkNN queries. Specifically, a user (a client) sends requests for an MkNN query based on consecutive rectangles, and the LSP (the server) returns k nearest neighbors (NNs) for any possible point in the rectangle. We show how to compute the consecutive rectangles and how to find the k NNs for these rectangles so that the user's trajectory remains private.
- We combat privacy threats in MkNN queries by requesting a higher confidence level or a higher number of data objects than required, or by combining both strategies.
- We improve the efficiency of the algorithm for the LSP to find k NNs for a rectangle with a user-customizable confidence level by exploiting different geometric properties. Our algorithm can compute both exact and approximate kNN answers.
- We present an extensive experimental study to demonstrate the efficiency and effectiveness of our approach. Our algorithm for the LSP is at least two times faster than the state-of-the-art.

The remainder of the paper is organized as follows. Section 2 discusses the problem setup and Section 3 reviews existing work. In Section 4, we give a overview of our system and in Section 5, we introduce the concept of confidence level. Sections 6 and 7 present our algorithms to request and evaluate a PMkNN query, respectively. Section 8 reports our experimental results and Section 9 concludes the paper.

2. Problem Formulation

A moving kNN (MkNN) query is defined as follows.

Definition 2.1. ***(MkNN query)*** *Let D denote a set of data objects in a two dimensional database, q the moving query point, and k a positive integer. An MkNN query returns for every position of q, a set A that consists of k data objects whose distances from q are less or equal to those of the data objects in $D - A$.*

A private *static* kNN query protects a user's privacy while processing a kNN query. Traditionally for private static kNN queries, the user requests k NNs[1] to the LSP with a rectangle that includes her current position [5, 9, 10]. Since the LSP does not know the user's actual location, it returns the k nearest data objects for every point of the rectangle. The straightforward application of private static kNN queries for processing an MkNN query cannot protect a user's *location privacy*, which is explained below.

2.1. Threat model for MkNN queries

Applying private static kNN queries to a PMkNN query requires that the user (the moving query point) continuously updates her location as a rectangle to an LSP so that the kNN answers are ensured for every point of her trajectory. The LSP simply returns the k NNs for every point of her requested rectangle. Thus, the moving user already has the k NNs for every position in the current rectangle. Since an MkNN query provides answers for every point of the user's trajectory, the next request for a new rectangle can be issued at any point before the user leaves the current rectangle. We also know that in a private static kNN query, a rectangle includes the user's current location at the time of requesting the rectangle to the LSP. Therefore, a straightforward application of private static kNN queries for processing an MkNN query requires the overlap of consecutive rectangles as shown in Figure 1(a). These overlaps refine the user's locations within the disclosed rectangles to the LSP. In the worst case, a user can issue the next request for a new rectangle when the user moves to the boundary of the current rectangle to ensure the availability of kNN answers for every point of the user's trajectory in real time. Even in this worst case scenario, the consecutive rectangles need to overlap at least at a point, which is the user's current location. We define the above described privacy threat as the *overlapping rectangle (privacy) attack* and omit the word "privacy" when the context is clear.

Definition 2.2. ***(Overlapping rectangle (privacy) attack)*** *Let $\{R_1, R_2, ..., R_n\}$ be a set of n consecutive rectangles requested by a user to an LSP in an MkNN query, where R_w and R_{w+1} overlap for $1 \leq w < n$. Since a user's location lies in the rectangle at the time it is sent to the LSP and the moving user requires the k NNs for every position, the user's location is in $R_w \cap R_{w+1}$ at the time of sending R_{w+1}, and the user's trajectory intersects $R_w \cap R_{w+1}$. As $(R_w \cap R_{w+1}) \subset R_w, R_{w+1}$, the overlapping rectangle attack enables an LSP to render more precise locations of the user.*

[1] In this paper, we use NN and nearest data object interchangeably.

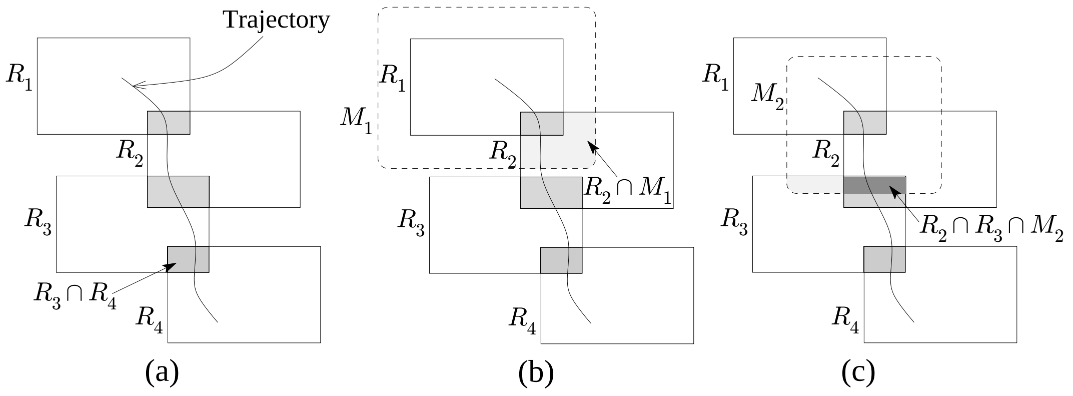


Figure 1: (a) Overlapping rectangle attack, (b) maximum movement bound attack, and (c) combined attack

Another attack is possible on a user's location privacy for MkNN queries if the user's maximum velocity is known. Existing research [5, 9, 12] has shown that if an LSP has rectangles from the same user at different times and the LSP knows the user's maximum velocity, then it is possible to refine a user's approximated location from the overlap of the current rectangle and the maximum movement bound with respect to the previous rectangle, called *maximum movement bound attack*. Figure 1(b) shows an example of this attack in an MkNN query that determines more precise location of a user in the overlap of R_2 and the maximum movement bound M_1 with respect to R_1 at the time of sending R_2.

For an MkNN query, the maximum movement bound attack is weaker than the overlapping rectangle attack as $(R_w \cap R_{w+1}) \subset (M_w \cap R_{w+1})$. However, the combination of these attacks can be stronger than each individual attack (see Figure 1(c)). In this example at the time of issuing R_3, the LSP derives M_2 from $R_1 \cap R_2$ rather than from R_2 and identifies the user's more precise location as $R_2 \cap R_3 \cap M_2$, where $(R_2 \cap R_3 \cap M_2) \subset (R_2 \cap R_3)$ and $(R_2 \cap R_3 \cap M_2) \subset (R_3 \cap M_2)$.

With the above described attacks, the LSP can progressively refine locations of a user. A ***private MkNN (PMkNN) query*** does not allow an LSP to infer more precise information than what the user reveals to the LSP for an MkNN query answers; thus, a PMkNN query prevents an LSP to identify more precise locations of the user within the provided rectangles by applying the overlapping rectangle attack and the maximum movement bound attack. Please note that there is no universally accepted view on what privacy protection implies for a user; it could either mean hiding a user's identity [7, 14, 15] or it could mean protecting privacy of the user's location while disclosing the user's identity [5, 16, 17] to the LSP. Therefore, a PMkNN query can also be defined based on both views. In this paper, we consider the second scenario where the user's location is unknown to the LSP since the user considers her location as private and sensitive information. In our privacy protection technique, a user can reveal her identity for personalized service. Moreover, nowadays there are many real world LBSs provided by Loopt [18] and Google's Android Market [19], where users need to identify themselves to access services.

A privacy protection technique that overcomes the overlapping rectangle attack and the maximum movement bound attack in an MkNN query needs to satisfy the following conditions.

Definition 2.3. ***(Conditions to overcome the overlapping rectangle attack and the maximum movement bound attack)***

1. *The user's location at the time of sending a rectangle cannot be refined to a subset of that rectangle.*
2. *The user's trajectory cannot be refined to a subset of the requested rectangle.*

A naïve solution to prevent overlapping rectangle attack is to request next rectangle after the user leaves the current rectangle. However, this solution cannot provide answers for the part of the trajectory between two rectangles and violates the definition of MkNN queries. Our proposed solution satisfies the two required conditions (see Definition 2.3) for every requested rectangle and provides kNN answers for every point of the user's trajectory. In our approach, a user does not need to send non-overlapping rectangles to prevent the overlapping rectangle attack. We show that our approach does not allow the LSP to refine the user's location or trajectory within the rectangle even if the user sends overlapping rectangles. Our approach also prevents the maximum movement bound attack based on the existing solutions [5, 9, 12] if the LSP knows the user's maximum velocity.

2.2. Attacker Model

In our proposed system, we assume that the attacker (i.e., the LSP) knows the user's identity, the sequence of rectangles and the requested query, which includes the number of data objects (i.e., k), the confidence level (i.e., the accuracy of answers) and the type of data objects (e.g., restaurant, gas station). The attacker applies the overlapping attack or the combined attack (if the maximum velocity is known) to refine the user's location within the provided rectangle. In our considered scenario, the user's location is sensitive and private data. There is no movement constraint for a user and the user can be anywhere within the rectangles: on a street, in a building or a boat. The attacker does not have any background knowledge about the user's location from any source that includes physical observation and mobile phone signal triangulation. We also assume that the attacker does not know the distribution of any revealed parameters including the confidence level and k.

2.3. Privacy Model

We will show that our proposed technique does not allow an LSP to refine a user's location within the provided rectangles using the overlapping rectangle attack and the maximum movement bound attack. We quantify location privacy based on what location information a user reveals to an LSP. For a PMkNN query, a user provides a sequence of rectangles, the confidence level and the number of requested data objects (i.e., k) to the LSP. Based on this revealed information, we measure location privacy as the (smallest) area to which an attacker can refine the trajectory (i.e., the sequence of user locations) relative to the data space. We call it *trajectory area* and define it in Section 8, as it requires concepts that are introduced later in the paper. The larger the trajectory area is, the higher is the user's location privacy since the lower is the probability that the user's location could be linked to a specific location. We also measure a user's location privacy by the number of requested rectangles per trajectory, i.e., the *frequency*. The smaller the number of requested rectangles for a fixed rectangle area, the less spatial constraints are available to the LSP for predicting the user's locations. Since there is no existing privacy model to measure location privacy for an MkNN query and such a model would require a rigorous mathematical treatment that is beyond the scope of this paper, we provide an approximate privacy model for the MkNN query. In the next step, we discuss the intuition behind modeling user privacy in terms of the trajectory area and frequency.

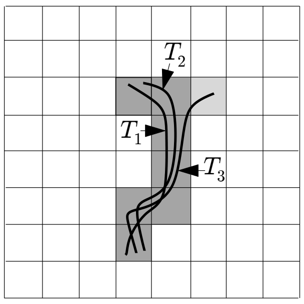


Figure 2: An example space divided into an 8×8 grid, where T_1 and T_2 are equivalent trajectories and T_3 is not equivalent to T_1 or T_2.

For ease of explanation, we make the following assumptions in our privacy model:

1. We initially assume the trajectory area to be a rectangle, although later we will show that the trajectory area can have different shapes. Our experiments will validate that our discussion for a rectangular trajectory area is also applicable for trajectory areas with different shapes.

2. We divide the total space into a grid, where each unit cell represents the minimum area that can be determined with a location device such as GPS. Figure 2 shows an example space, which is divided into 8×8 grid.

3. We assume that each trajectory intersects a cell once and trajectories that go through the same set of grid cells fall in the same equivalent class. For example, in Figure 2, trajectories T_1 and T_2 are considered as equivalent and trajectory T_3 is not equivalent to T_1 or T_2 because the starting grid cell is different for T_3.

In Section 8, we will show that the trajectory area depends on the number of requested rectangles (i.e., frequency), confidence level and the k nearest data objects. Although frequency is a parameter that affects the trajectory area, the aim of this discussion is to motivate that the trajectory area and frequency are two independent measures for location privacy. We discuss two cases separately: the effect of varying the trajectory area and frequency on a user's location privacy. For the first case, we assume that different trajectory areas can be obtained for a fixed frequency by varying confidence

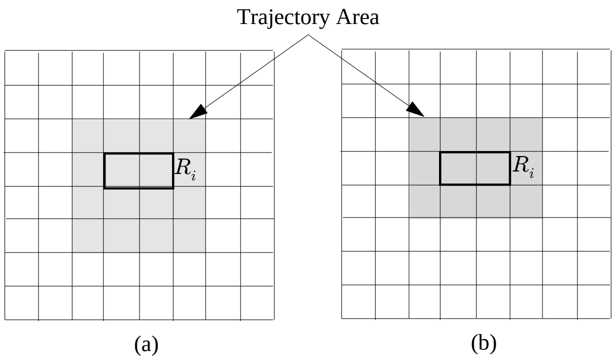


Figure 3: Varying trajectory area

level and/or k. Similarly, for the second case, the assumption of a fixed trajectory area for different frequencies can be achieved by varying confidence level and/or k.

Case (i): Varying the trajectory area for a fixed frequency. A larger trajectory area comprises more grid cells, which in turn increases the number of possible trajectories of the user available to the LSP. With the increase of the user's possible trajectories, the LSP becomes more uncertain about the user's locations within the trajectory area and thereby the user's location privacy is increased. Figure 3 shows an example, where the frequency is 1 and the rectangle area consists of 2 grid cells, i.e., the user is located at any of these 2 grid cells at the time of requesting the rectangle. The trajectory area in Figure 3(a) and Figure 3(b) are composed of 16 and 12 grid cells, respectively. Thus the user's location privacy in Figure 3(a) is higher than that of Figure 3(b).

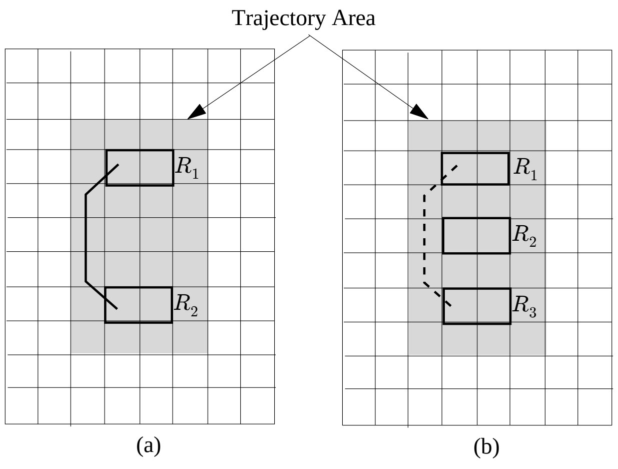


Figure 4: Varying frequency

Case (ii): Varying the frequency for a fixed trajectory area. The increase of frequency increases the number of available constraints (i.e., rectangles) for a user's trajectory to the LSP and thus lowers the user's location privacy. Figure 4 shows an example, where the trajectory area is same in both scenarios but the frequencies are different. Since the trajectory area is same, the set of possible trajectories of the user determined from the trajectory area are also same. However, the LSP can refine the set of possible trajectories from the available rectangles of the user. In Figure 4(a), since the user's trajectory goes through the rectangle R_1 and R_2, the LSP can eliminate the trajectories from the set that do not go through R_1 or R_2 and thereby reduce the number of possible trajectories for the user. In Figure 4(b), since the user's trajectory goes through rectangles R_1, R_2, and R_3, the LSP can further refine the set of trajectories computed for the scenario in Figure 4(a); the LSP eliminates the trajectories from the set that do not go through R_3 in addition to R_1 and R_2.

As an example, if we consider trajectories of length of 5 units, then the number of possible trajectories are 68 and 32 in Figure 4(a) and Figure 4(b), respectively. Figure 4(a) shows one of the possible 68 trajectories, which is not included in the set of 32 trajectories for the scenario in Figure 4(b) because the trajectory does not go through R_2 in Figure 4(b). Thus, the number of possible trajectories for the user decreases with the increase of the available constraints to the LSP, i.e., the user's location privacy decreases with the increase of the frequency.

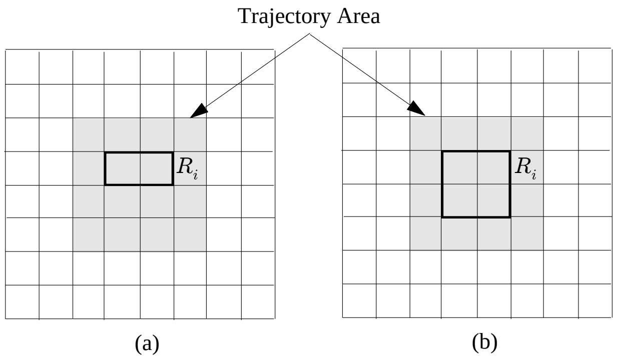


Figure 5: Different rectangle areas

In the above mentioned example scenario, we compare frequencies for a fixed rectangle area. The areas of the requested rectangles used for a PMkNN query can also be different. A user's location privacy increases with the increase of the rectangle area for a fixed frequency because the larger rectangle represents more relaxed constraint to the LSP. For example, in Figure 5(a), the set of possible trajectories of the user needs to go through any of 2 grid cells of the rectangle, whereas in Figure 5(b), the set of possible user trajectories needs to go through any of 4 grid cells of the rectangle. Thus the set of possible trajectories of the user is smaller in the scenario of Figure 5(a) than that of Figure 5(b). We know that the smaller set of trajectories represents a lower location privacy for the user as the probability of the LSP to identify the user's locations within the trajectory area increases.

It is important to note that our proposed privacy model is generic and applicable to any other privacy preserving approaches that can compute trajectory area and frequency. Our privacy model is generalized, i.e., the privacy model is independent of the algorithm used to measure trajectory area and frequency.

3. Related Work

Privacy protection techniques. Most research on user privacy in LBSs has focused on static location-based queries that include nearest neighbor queries [7, 14, 15, 16, 20, 21], group nearest neighbor queries [22] and proximity services [23]. Different strategies such as K-anonymity, obfuscation, l-diversity, and cryptography have been proposed to protect the privacy of users.

K-anonymity techniques (e.g., [8, 14]) make a user's identity indistinguishable within a group of K users. Obfuscation techniques (e.g., [16, 24]) degrade the quality of a user's location by revealing an imprecise or inaccurate location and l-diversity techniques (e.g., [6, 10]) ensure that the user's location is indistinguishable from $l - 1$ other locations. Both obfuscation, and l-diversity techniques focus on hiding the user's location from the LSP instead of the identity. Cryptographic techniques (e.g., [25, 26]) allow users to access LBSs without revealing their locations to the LSP, however, these techniques incur cryptographic overhead.

K-anonymity, obfuscation, or l-diversity based approaches for private static queries cannot protect privacy of users for continuous LBSs because they consider each request of a continuous query as an independent event, i.e., the correlation among the subsequent requests is not taken into account. Recently different approaches [9, 5, 11, 12, 13, 27, 28, 29] have been proposed to address this issue.

The authors in K-anonymity based approaches [11, 13, 27, 28] for continuous queries focus on the privacy threat on a user's identity that arises from the intersection of different sets of K users involved in the consecutive requests of a continuous query. Since we focus on how to hide a user's trajectory while disclosing the user's identity to the LSP, these approaches are not applicable for our purpose. On the other hand, existing obfuscation and l-diversity based approaches [9, 5, 12] for continuous queries have only addressed the threat of the maximum movement bound attack. However, none of these approaches have identified the threat on location privacy that arises from the overlap of consecutive regions (e.g., rectangles). The trajectory anonymization technique proposed in [29] assumes that a user knows her trajectory in advance for which an LBS is required, whereas other approaches including ours consider an unknown future trajectory of the user.

Though cryptographic approaches [25, 26] offer strongest location privacy by not revealing user locations to the LSP, these approaches incur cryptographic overheads and cannot use existing spatial indexing techniques to store data on the server. In [26], the data space is encrypted from two dimension to one dimension space using Hilbert curves by a trusted third party and the transformed space can be decrypted using a key, which is not known to the LSP and the users. This approach assumes that the user's tamper-resistant device stores the key and encrypts the location before forwarding a query to the LSP. The LSP evaluates the kNN query in the transformed space and returns the encrypted data objects which are again decrypted to the original space (two dimensional space) in the user's device. Since the query for k nearest data objects is evaluated in an encrypted space which might not always preserve the proximity relations of the original space, the returned data objects may not be the actual answers. In this approach, the user privacy could be violated if any of these tamper-resistant devices gets compromised. On the other hand, private information retrieval (PIR) protocols have been proposed [25] to retrieve the approximate and exact nearest data objects without disclosing a user's location. In this approach, the space is divided into grid cells and the data objects associated with each cell are stored in a format required by PIR protocols. The user provides her encrypted cell, where she is located, using PIR, and determines the nearest data object from the retrieved encrypted data objects. Although this approach ensures privacy for both static and continuous queries, it incurs a high pre-processing overhead compared to spatial cloaking [30]. Moreover, this approach only supports queries for the nearest data object; the extension to k nearest data objects is not straightforward and requires to maintain a separate data storage for every k, which in turn leads to high computational and storage overheads.

In the literature there are other privacy preserving concepts such as t-closeness [31] and differential privacy [32, 33, 34] that are used to protect user privacy while publishing user data. The published data allows organizations or researchers to perform useful analyses for many applications that include urban planning, traffic monitoring, and mining human behavior. In this scenario, the user data is stored on a database and is modified before shared with others so that both user privacy and data utility are maintained. For example, the concept of differential privacy has been proposed to address the problem of sharing data for statistical analyses without allowing others to identify the data of an individual from the revealed information. On the other hand, in this paper, we have focused on the problem of protecting a user's privacy by controlling the release of the user's location information to the LSP while accessing an MkNN query. Therefore our studied problem is orthogonal to the problem of privacy preserving data publishing and concepts like t-closeness or differential privacy are not applicable to our problem.

***K*NN algorithms.** To provide the query answers to the user, the LSP needs an algorithm to evaluate a kNN query for the user's location. *Depth first search* (DFS) [35] and *best first search* (BFS) [36] are two well known algorithms to find the k NNs with respect to a point using an R-tree [37]. If the value of k is unknown, e.g., for an incremental kNN query, the next set of NNs can be determined with BFS. We use BFS in our proposed algorithm to evaluate a kNN query with respect to a rectangle. The BFS starts the search from the root of the R-tree and stores the child nodes in a priority queue. The priority queue is ordered based on the minimum distance between the query point and the *minimum bounding rectangles* (MBRs) of R-tree nodes or data objects. In the next step, it removes an element from the queue, where the element is the node representing the MBR with the minimum distance from the query point. Then the algorithm again stores the child nodes or data objects of the removed node on the priority queue. The process continues until k data objects are removed from the queue. When the objects are moving, Zhang et. al. show how to evaluate kNN queries [38] and join queries [39, 40] based on the TPR-tree [41].

Researchers have also focused on developing algorithms [12, 14, 21, 42, 43, 44] for evaluating a kNN query for a user's imprecise location such as a rectangle or a circle. In [43], the authors have proposed an approximation algorithm that ensures that the answer set contains one of the k NNs for every point of a rectangle. The limitation of their approximation is that users do not know how much more they need to travel with respect to the actual NN, i.e., the accuracy of answers. Our algorithm allows users to specify the accuracy of answers using a confidence level.

To prevent the overlapping rectangle attack, our proposed approach requires a kNN algorithm that returns a candidate answer set including all data objects of a region in addition to the k NNs with respect to every point of a user's imprecise location. The availability of all data objects for a *known region* to the user in combination with the concept of hiding the user's required confidence level and the required number of NNs from the LSP can prevent the overlapping rectangle attack (see Section 6). Among all existing kNN algorithms for a user's imprecise location [12, 14, 21, 42, 43, 44], only Casper [14] supports a known region; the algorithm returns all data objects of a *rectangular region* (i.e., the known region) that include the NNs with respect to a rectangle. However, Casper can only work for NN queries and it is not straightforward to extend Casper for $k > 1$. Thus, even if Casper is modified to incorporate the confidence level concept, it can only support PMkNN queries for $k = 1$.

Moreover, for a single nearest neighbor query, Casper needs to perform on the database multiple searches, which incur high computational overhead. Casper executes four individual single nearest neighbor queries with respect to four corner points of the rectangle. Then using these neighbors as filters, Casper expands the rectangle in all directions to compute a range that contains the NNs with respect to all points of the rectangle. Finally, Casper has to again execute a range query to retrieve the candidate answer set. We propose an efficient algorithm that finds the kNNs with a specified confidence level for a rectangle in a single search.

4. System Overview

Our approach for PMkNN queries is based on a client-server paradigm, which is a common architecture for providing LBSs. A client is a moving user who sends a PMkNN query request via a mobile device and the server is the LSP that processes the query. We assume that the user's mobile device is equipped with positioning technologies (e.g., GPS or assisted GPS) and can communicate with the LSP through the Internet or telecommunication networks. The moving user sends her imprecise location as a rectangle to the LSP, which we call *obfuscation rectangle* in the remainder of this paper.

We introduce the parameter confidence level, which provides a user with an option to trade the accuracy of the query answers for location privacy. Intuitively, the confidence level of the user for a data object guarantees that the distance of the data object to the user's location is within a bound of the actual nearest data object's distance. In Section 5, we formally define and show how a user and an LSP can compute the confidence level for a data object.

In our system, a user does not reveal the required confidence level and the required number of NNs to the LSP while requesting a PM*k*NN query; instead the user *specifies higher values than the required ones*. This allows the user to have the required number of NNs with the required confidence level for an additional part of her trajectory, which is unknown to the LSP. As the user has the required answers for an additional part of her trajectory, the consecutive rectangles do not have to always overlap. Even if the rectangles overlap, there is no guarantee that the user is located in the overlap at the time of requesting the rectangle and the user's trajectory passes through the overlap. Thus the LSP cannot apply the overlapping rectangle attack by correlating the user's current obfuscation rectangle with the previous one.

Note that requesting a higher confidence level (*cl*) and a higher number of nearest data objects (*k*) than required are two different strategies to counter the overlapping rectangle attack. If a user is willing to travel slightly more then the user can opt for a higher confidence level. If a user is willing to pay more for the communication overhead due to transferring a large number of data objects then the user may prefer to request a higher number of nearest data objects. A user can also combine both strategies to increase the level of user privacy.

In summary, in our approach for a PM*k*NN query, a moving user needs to continuously send queries to the LSP, where each query contains the user's current obfuscation rectangle R_w, a specified confidence level cl and a specified number of nearest data objects k. For every requested query of the user, the LSP returns a set of candidate data objects P that include k nearest data objects with the specified confidence level cl for R_w. Note that, when an LSP finds *k*NNs with a confidence level cl for an obfuscation rectangle, the LSP returns a set of data objects that include k nearest data objects for every point of the obfuscation rectangle with a confidence level of at least cl. Thus, the moving user can delay her next query to the LSP as long as the user has the required number of nearest data objects with the required confidence level. Algorithm 1 details the handshake between a user and an LSP during the access of a PM*k*NN query. To ensure real time availability of the answer, the moving user can also send the next query in advance instead of delaying as long as the user has the required M*k*NN answers. We use a parameter, called *safe distance* (defined in Section 6), to determine when to compute the next obfuscation rectangle and to send the new query to the LSP.

Algorithm 1: HANDSHAKING_USER_LSP

```
1.1 while service required do
1.2     User → LSP: Send($R_w$, $cl$, $k$)
1.3     LSP → User: Send($P$)
1.4     Delay as long as the user has the required
        answers
```

In Section 6, we present a technique to compute a user's consecutive obfuscation rectangles for requesting a PM*k*NN query. The first obfuscation rectangle can be computed according to a user's privacy requirement using any existing techniques for static queries (e.g., [20]). For the computation of subsequent obfuscation rectangles of the user, our technique ensures that the LSP cannot refine the user's location within the obfuscation rectangles by applying the overlapping rectangle attack and the maximum movement bound attack. Another important advantage of our technique is that for the computation of the consecutive obfuscation rectangles, the user does not need to trust any other party such as an intermediary trusted server [14].

An essential component of our approach for a PM*k*NN query is an algorithm for the LSP that finds the specified number of NNs for the obfuscation rectangle with the specified confidence level. In Section 7, we exploit different properties of the confidence level with respect to an obfuscation rectangle to develop an efficient algorithm in a single traversal of the *R*-tree.

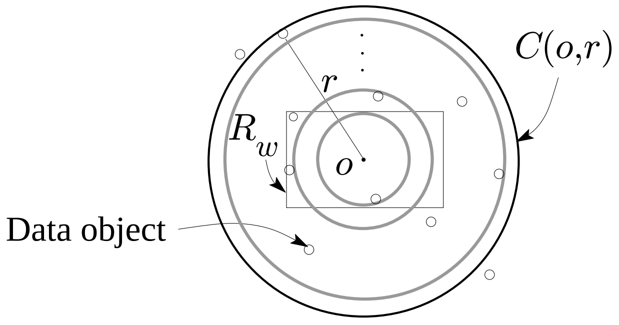


Figure 6: Known Region

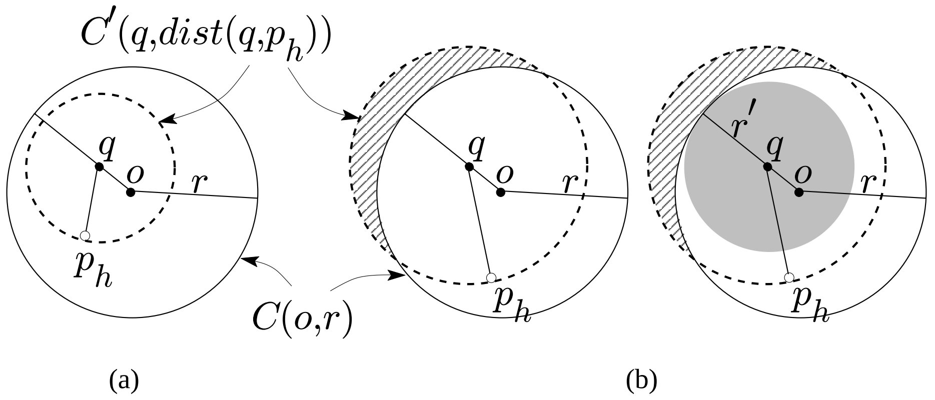


Figure 7: Confidence Level

5. Confidence Level

The confidence level measures the accuracy for a nearest data object with respect to a user's location. If the confidence level for the k nearest data objects is 1 then they are the actual k NNs. A confidence level less than 1 provides a worst case bound on the distance compared to the actual k^{th} nearest data object, e.g., for a nearest data object with a confidence level of 0.5 the user has to travel at most twice the distance compared to the actual NN.

A user's confidence level for any nearest data object is determined by other data objects' locations surrounding the user's location. The region where the location of all data objects are known is called the *known region*. The user measures the confidence level to determine when to send the next query and the LSP measures the confidence level to determine the query answer. We first show how an LSP and a user compute the known region, and then discuss the confidence level.

5.1. Computing a known region

Suppose a user provides an obfuscation rectangle R_w for any positive integer w, to the LSP while requesting a PMkNN query. For the ease of explanation, we assume initially that the user specifies a confidence level of 1, i.e., the returned answer set includes the actual kNN answers for R_w. To evaluate kNN answers, the LSP starts a best first search (BFS) using the center o of R_w as the query point and incrementally finds the next NN from o until the k NNs are discovered for all points of R_w.

The search region covered by BFS is at any execution stage a circular region $C(o,r)$, where the center o is the center of R_w and the radius r is the distance between o and the last discovered data object (see Figure 6). Since the locations of all data objects in $C(o,r)$ are already discovered, $C(o,r)$ is the known region for the LSP. The LSP returns all data objects in $C(o,r)$ to the user, although some of them might not be the k NNs for any point of R_w. This enables the user to have $C(o,r)$ as the known region.

5.2. Measuring the confidence level

Since the confidence level can have any value in the range (0,1], we remove our assumption of a fixed confidence level of 1. In our approach, the known region $C(o,r)$ is used to measure the confidence level. Without loss of generality, let p_h be the nearest data object among all data objects in $C(o,r)$ from a given location q, where h is an index to name the data objects and let $dist(q,p_h)$ represent the Euclidean distance between q and p_h. There are two possible scenarios based on different positions of p_h and q in $C(o,r)$. Figure 7(a) shows a case where the circular region $C'(q,dist(q,p_h))$ centered at q with radius $dist(q,p_h)$ is within $C(o,r)$. Since p_h is the nearest data object from q within $C(o,r)$, no other data object can be located within $C'(q,dist(q,p_h))$. This case provides the user at q with a confidence level 1 for p_h. However, $C'(q,dist(q,p_h))$ might not be always completely within the known region. Figure 7(b)(left) shows such a case, where a part of $C'(q,dist(q,p_h))$ falls outside $C(o,r)$ and as the locations of data objects outside $C(o,r)$ are not known, there might be some data objects located in the part of $C'(q,dist(q,p_h))$ outside $C(o,r)$ that have a smaller distance than p_h from q. Since p_h is the nearest data object from q within $C(o,r)$, there is no data object within distance r' from q (Figure 7(b)(right)), where r' is the radius of the maximum circular region within $C(o,r)$ centered at q. But there might be other data objects within a fixed distance d_f from q, where $r' < d_f \leq dist(q,p_h)$. In this case the

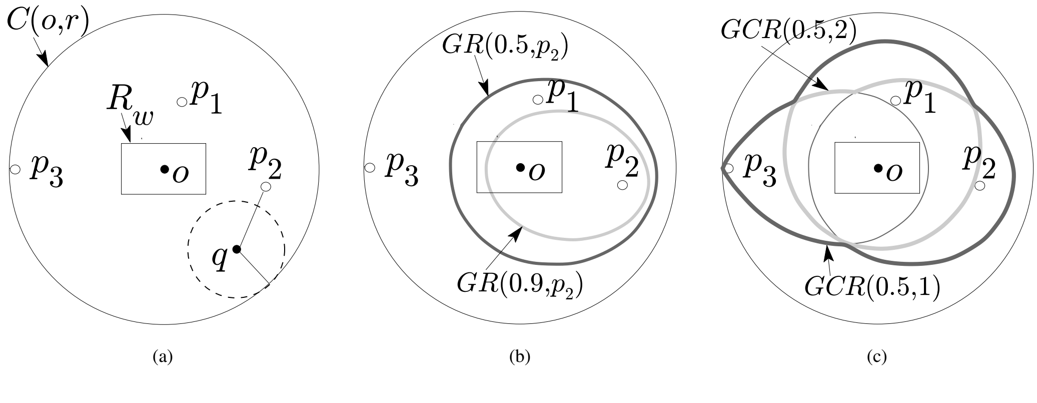


Figure 8: (a) $CL(q, p_2)$, (b) $GR(cl, p_2)$ and (c) $GCR(cl, k)$

confidence level of the user at q for p_h is less than 1. On the other hand, if q is outside of $C(o,r)$ then the confidence level of the user at q for p_h is 0 because r' is 0. We define the confidence level of a user located at q for p_h in the more general case, where p_h is redefined as any of the nearest data object in $C(o,r)$ instead of the nearest data object among all data objects in $C(o,r)$.

Definition 5.1. ***(Confidence level)*** *Let $C(o,r)$ be the known region, P the set of data objects in $C(o,r)$, q the point location of a user, p_h the j^{th} nearest data object in P from q for $1 \le j \le |P|$. The distance r' represents the radius of the maximum circular region within $C(o,r)$ centered at q. The confidence level of the user located at q for p_h, $CL(q,p_h)$, can be expressed as:*

$$CL(q,p_h) := \begin{cases} 0 & \textit{if } q \notin C(o,r) \\ 1 & \textit{if } q \in C(o,r) \wedge dist(q,p_h) \le r' \\ \frac{r'}{dist(q,p_h)} & \textit{otherwise.} \end{cases}$$

Since we focus on NN queries, we use distance instead of area as the metric for the confidence level. A distance-based metric ensures that there is no other data object within a fixed distance from the position of a user. Thus, the distance-based metric is a measure of accuracy for a data object to be the nearest one. An area-based metric, however, that is based on the area of $C'(q, dist(q,p_h))$ intersecting with $C(o,r)$ could only express the likelihood of an data object to be the nearest one. Thus, an area-based metric cannot measure the accuracy of the data object to be the nearest one.

6. Client-side Processing

We present a technique for computing consecutive obfuscation rectangles to request a PM*k*NN query, where the LSP cannot apply the overlapping rectangle attack to approximate the user's location. Suppose a user requests an obfuscation rectangle R_w and a confidence level cl at any stage of accessing the PM*k*NN query. Since a user can be anywhere in R_w, the LSP returns P, the set of data objects in the known region $C(o,r)$, that includes the k NNs with a confidence level of at least cl for every point of R_w. Having $C(o,r)$, a user can compute the confidence level for the k NNs even from outside of R_w.

Although some data objects in P might not be the k NNs for any point of R_w, they might be k NNs for a point outside R_w with a confidence level of at least cl. In addition, some data objects, which are the k NNs for some portions of R_w, can be also the k NNs from locations outside of R_w with a confidence level of at least cl. For example for $cl = 0.5$ and $k = 1$, Figure 8(a) shows that a point q, located outside R_w, has a confidence level[2] greater than 0.5 for its nearest data object p_2. On the other hand, from a data object's viewpoint, Figure 8(b) shows two regions surrounding a data object p_2, where for any point inside these regions a user has a confidence level of at least 0.90, and 0.50, respectively for p_2[3]. We call such a region *guaranteed region*, $GR(cl, p_h)$ with respect to a data object p_h for a specific confidence level cl. We define $GR(cl, p_h)$ as follows.

Definition 6.1. ***(Guaranteed region)*** *Let $C(o,r)$ be the known region, P the set of data objects in $C(o,r)$, p_h a data object in P, and cl the confidence level. The guaranteed region with respect to p_h, $GR(cl,p_h)$, is the set*

[2]The confidence level of any point represents the confidence level of a user located at that point.

[3]Note that, whenever we mention the confidence level of a point for a data object then the data object can be any of the j^{th} NN from that point, where $1 \le j \le |P|$.

of all points such that $\{CL(q, p_h) \geq cl\}$ *for any point* $q \in GR(cl, p_h)$.

From the guaranteed region of every data object in P we compute the *guaranteed combined region*, $GCR(cl, k)$, where a user has at least k data objects with a confidence level of at least cl. Figure 8(c) shows an example, where $P = \{p_1, p_2, p_3\}$ and $cl = 0.5$. Then for $k = 1$, the black bold line shows the boundary of $GCR(0.5, 1)$, which is the union of $GR(0.5, p_1)$, $GR(0.5, p_2)$ and $GR(0.5, p_3)$. For $k = 2$, the ash bold line shows the boundary of $GCR(0.5, 2)$, which is the union of $GR(0.5, p_1) \cap GR(0.5, p_2)$, $GR(0.5, p_2) \cap GR(0.5, p_3)$ and $GR(0.5, p_3) \cap GR(0.5, p_1)$. We define $GCR(cl, k)$ as follows.

Definition 6.2. *(Guaranteed combined region) Let* $C(o, r)$ *be the known region,* P *the set of data objects in* $C(o, r)$*,* p_h *a data object in* P*,* cl *the confidence level,* k *the number of data objects, and* $GR(cl, p_h)$ *the guaranteed region. The guaranteed combined region,* $GCR(cl, k)$*, is the union of the regions where at least* k $GR(p_h, cl)$ *overlap, i.e.,* $\cup_{P' \subseteq P \wedge \|P'\| = k} \{\cap_{h \in P'} GR(p_h, cl)\}$.

According to Definition 5.1, the user has higher a confidence level for closer data objects, which implies Lemma 6.1:

Lemma 6.1. *If the confidence level of a user located at* q *is at least* cl *for any* k *data objects, then the confidence level of the user is also at least* cl *for the* k *NNs from* q.

Since for any point in $GCR(cl, k)$, a user has at least k data objects with a confidence level of at least cl, Lemma 6.1 shows that for any point in $GCR(cl, k)$ the user also has the k NNs with a confidence level of at least cl. Thus, in our technique, the moving user can use the retrieved data objects from the outside of R_w and delay the next obfuscation rectangle R_{w+1} until the user leaves $GCR(cl, k)$. Although delaying R_{w+1} in this way may allow a user to avoid an overlap of R_w and R_{w+1}, the threat to location privacy is still in place. Since the LSP can also compute $GCR(cl, k)$, similar to the overlapping rectangle attack, the user's location can be computed more precisely by the LSP from the overlap of R_{w+1} and current $GCR(cl, k)$ (see Figure 9(a) for $GCR(0.5, k) \cap R_{w+1}$).

To overcome the above mentioned attack and the overlapping rectangle attack, the key idea of our technique is *to increase the size of GCR without informing the LSP about this extended region*. To achieve this the user has three options during a PM*k*NN query: the user specifies a higher value than (i) the required confidence level or (ii) the required number of NNs or (iii) both. Let cl_r and k_r represent the required confidence level and the required number of NNs for a user (both are not revealed to the LSP), respectively, and cl and k represent the specified confidence level and the specified number of NNs to the LSP by the user, respectively.

Case (i): a user specifies a higher value than the required confidence level, i.e., $cl > cl_r$**.** We know that the GCR is constructed from GRs of data objects in P and the GR of a data object becomes smaller with the increase of the confidence level for a fixed $C(o, r)$ as shown in Figure 8(b), which justifies the following lemma.

Lemma 6.2. *Let* $cl > cl_r$ *and* $k = k_r$*. Then* $GCR(cl_r, k_r) \supset GCR(cl, k)$ *for a fixed* $C(o, r)$.

Since $GCR(cl_r, k_r) \supset GCR(cl, k)$, now the user can delay the next obfuscation rectangle R_{w+1} until the user leaves $GCR(cl_r, k_r)$. Since the LSP does not know $GCR(cl_r, k_r)$, it is not possible for the LSP to find more precise location from the overlap of $GCR(cl_r, k_r)$ and R_{w+1}. Figure 9(b) shows an example for $k = 1$, where a user's required confidence level is $cl_r = 0.5$ and the specified confidence level is $cl = 0.9$. The LSP does not know about the boundary of $GCR(0.5, 1)$ and thus cannot find the user's precise location from the overlap of $GCR(0.5, 1)$ and R_{w+1}.

However, R_{w+1} has to be in $C(o, r)$ of R_w. Otherwise, the LSP is able to determine more precise location of the user as $R_{w+1} \cap C(o, r)$ at the time of requesting R_{w+1}. For any location outside $C(o, r)$, the user has a confidence level 0 which in turn means that the user's location cannot be within the region of R_{w+1} that falls outside $C(o, r)$ at the time of requesting R_{w+1}. As a result whenever $C(o, r)$ is small, then the restriction might cause a large part of R_{w+1} to overlap with $GCR(cl, k)$ and R_w.

The advantage of our technique is that this overlap does not cause any privacy threat for the user's trajectory due to the availability of $GCR(cl_r, k_r)$ to the user. Since there is no guarantee that the user's trajectory passes through the overlap or not, the LSP is not able to determine the user's precise trajectory path from the overlap of R_{w+1} with $GCR(cl, k)$ and R_w. Without loss of generality, Figures 9(c) and 9(d) show two examples, where R_{w+1} overlaps with $GCR(0.9, 1)$ for $cl_r = 0.5$, $cl = 0.9$, and $k = 1$. In Figure 9(c) we see that the trajectory does not pass through $GCR(0.9, 1) \cap R_{w+1}$, whereas Figure 9(d) shows a case, where the trajectory passes through the overlap.

Note that the distribution of data objects does not result in any attack that refines a user's more precise location within the obfuscation rectangle. If the distribution

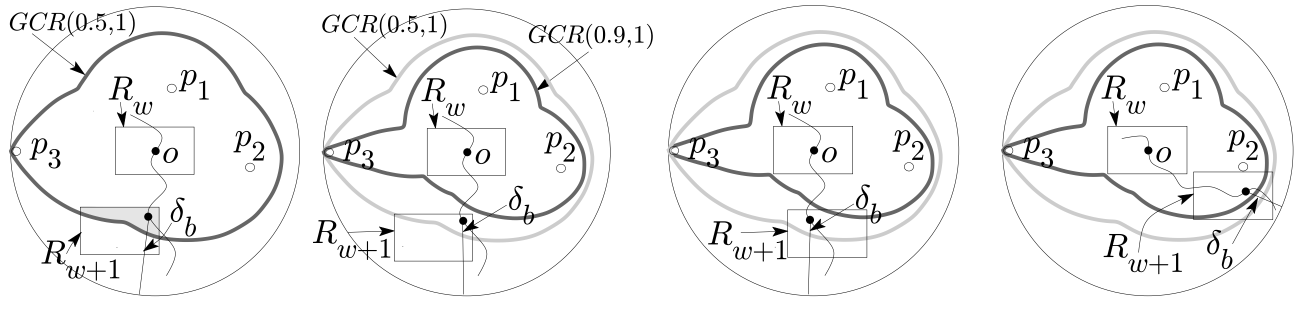


Figure 9: (a) An attack from $R_{w+1} \cap GCR(0.5, 1)$, (b)-(d) Removal of attacks with $cl_r = 0.5$ and $cl = 0.9$

of data objects is dense around a user's location then the known region becomes smaller. Although the smaller known region may cause the current obfuscation rectangle to be closer to the previous obfuscation rectangle or even to overlap with the previous one, the LSP can only approximate the user's location as the obfuscation rectangle. Our technique does not allow the LSP to further refine the user's location in the obfuscation rectangle from the overlap. However, the dense distribution of data objects may lower the level of a user's location privacy, which we measure in terms of trajectory area and frequency, due to the smaller known region.

Case (ii): a user specifies a higher value than the required number of NNs, i.e., $k > k_r$. From the construction of a *GCR*, we know that $GCR(cl, k + 1) \subset GCR(cl, k)$ for a fixed $C(o, r)$, which leads to the following lemma.

Lemma 6.3. *Let $cl = cl_r$ and $k > k_r$. Then $GCR(cl_r, k_r) \supset GCR(cl, k)$ for a fixed $C(o, r)$.*

Since we also have $GCR(cl_r, k_r) \supset GCR(cl, k)$ for the second option, similar to the first option, a user can protect her location privacy using the extended region, which is used when the user cannot sacrifice the accuracy of answers.

Case (iii): a user requests higher values for both confidence level and the number of NNs than required. The user can obtain a larger extension for the $GCR(cl_r, k_r)$ as both cl and k contribute to extend the region. The extension reduces the number of obfuscation rectangles sent to the LSP, which increases a user's location privacy since the LSP has less information about the user locations. The location privacy also increases with the increase of the difference between cl and cl_r or k and k_r because with the decrease of cl_r or k_r, the size of $GCR(cl_r, k_r)$ increases for a fixed $C(o, r)$ and with the increase of cl or k, $C(o, r)$ becomes larger, which results in a larger $GCR(cl_r, k_r)$.

In our technique, a user has different choices: a user can delay R_{w+1} until her trajectory intersects with the boundary of $GCR(cl_r, k_r)$; or the user sends R_{w+1} in advance to ensure the local availability (to the user) of k_r NNs with a confidence level of at least cl_r for a part of her future trajectory and thereby avoiding any delay from the LSP. Since the future trajectory is unknown, our algorithm always ensures the local availability of query answers at least for a user specified distance in all directions from the user's current position. To ensure the local availability of query answers in real time, we use two variables:

- *Boundary distance* δ_b: the minimum distance of user's current position q from the boundary of $C(o, r)$.
- *Safe distance* δ: the user specified distance, which is used to determine when the next request needs to be sent.

Thus, the next request is sent to the LSP when $\delta_b \leq \delta$.

The parameters cl, cl_r, k, k_r, δ, and obfuscation rectangle area can be changed according to the user's privacy profile and quality of service requirements. Different cl, cl_r, k, k_r, and δ in consecutive requests prevent an LSP from gradually learning or guessing any bound of cl_r and k_r to predict a more precise user location.

Based on the above discussion of our technique, we present the algorithm that protects the user's location privacy while processing an M*k*NN query. algorithm, we summarize commonly used symbols in Table 1.

6.1. Algorithm

Algorithm 2, Request_PM*k*NN, shows the steps for requesting a PM*k*NN query. A user initiates the query with an obfuscation rectangle R_w that includes her current location q. The parameters cl, cl_r, k, k_r, and δ are set according to the user's requirement. Then a request is sent with R_w to the LSP for k NNs with a confidence

Symbol	Meaning
R_w	Obfuscation Rectangle
cl_r	Required confidence level
cl	Specified confidence level
k_r	Required number of NNs
k	Specified number of NNs
$C(o,r)$	Known region
$GCR(.,.)$	Guaranteed combined region
δ	Safe distance
δ_b	Boundary distance

Table 1: Symbols

level cl. The LSP returns the set of data objects P that includes the k NNs for every point of R_w with a confidence level of at least cl. According to Lemma 6.1, the user has the k_r NNs with a confidence level of at least cl_r as long as the user resides within $GCR(cl_r, k_r)$. Maintaining the rank of k_r NNs from P for every position of the user's trajectory is an orthogonal problem for which we can use any existing approaches (e.g., [45]).

Algorithm 2: REQUEST_PM*k*NN

```
2.1  $w \leftarrow 1$
2.2  $cl, cl_r \leftarrow$ user specified and required confidence level
2.3  $k, k_r \leftarrow$ user specified and required number of NNs
2.4  $\delta \leftarrow$ user specified safe distance
2.5  $R_w \leftarrow GenerateRectangle(q)$
2.6  $P \leftarrow RequestkNN(R_w, cl, k)$
2.7  while service required do
2.8      $q \leftarrow NextLocationUpdate()$
2.9      $p_{hk} \leftarrow {k_r}^{th}$ NN from $q$
2.10     $cl, cl_r \leftarrow$ user specified and required confidence level
2.11     $k, k_r \leftarrow$ user specified and required number of NNs
2.12     $\delta \leftarrow$ user specified safe distance
2.13     $\delta_b \leftarrow r - dist(o, q)$
2.14     if $(r \le cl_r \times dist(p_{hk}, q) + dist(o, q))$ or $(\delta_b \le \delta)$ then
2.15         $R_{w+1} \leftarrow GenerateRectangle(q, C(o, r))$
2.16         $P \leftarrow RequestkNN(R_{w+1}, cl, k)$
2.17         $w \leftarrow w + 1$
```

For every location update, the algorithm checks two conditions: whether the user's current position q is outside her current $GCR(cl_r, k_r)$ or the minimum boundary distance from $C(o,r)$, δ_b, has become less or equal to the user specified distance, δ. To check whether the user is outside her $GCR(cl_r, k_r)$, the algorithm checks the constraint $r \le cl_r \times dist(p_{hk}, q) + dist(o, q)$, where r is the radius of current known region and $cl_r \times dist(p_{hk}, q) + dist(o, q)$ represents the required radius of the known region to have k_r NNs with a confidence level of at least cl_r from the current position q. Note that the required radius $cl_r \times dist(p_{hk}, q) + dist(o, q)$ is computed using Definition 5.1. For the second condition, δ_b is computed by subtracting $dist(o, q)$ from r (Line 2.13). If any of the two conditions in Line 2.14 becomes true, then the new obfuscation rectangle R_{w+1} is computed with the restriction that it must be included within the current $C(o, r)$. The function *GenerateRectangle* is used to compute an obfuscation rectangle within $C(o, r)$ according to the user's specified area. After computing R_{w+1}, the next request is sent and k NNs are retrieved for R_{w+1} with a confidence level of at least cl. The process continues as long as the service is required.

The function *GenerateRectangle* is used to compute an obfuscation rectangle for a user according to her privacy requirement. The obfuscation rectangle can be computed using the obfuscation technique proposed in [20].

Note that, Algorithm 2 to protect a user's location privacy for an M*k*NN query with obfuscation rectangles can be also generalized for the case where a user uses other geometric shapes (e.g., a circle) to represent the imprecise locations if the known region for other geometric shapes is also a circle. For example, if a user uses obfuscation circles instead of obfuscation rectangles then the overlapping rectangle attack turns into overlapping circle attack. From Algorithm 2, we observe that our technique to protect overlapping rectangle attack is independent of any parameter of obfuscation rectangle; it only depends on the center and radius of the known region. Thus, as long as the representation of the known region is a circle, our technique can be also applied for an overlapping circle attack.

The following theorem shows the correctness of the algorithm REQUEST_PM*k*NN.

Theorem 6.1. *The algorithm* REQUEST_PM*k*NN *protects a user's location privacy for MkNN queries.*

PROOF. The obfuscation rectangles R_{w+1} for a user requesting a PM*k*NN query always overlaps with $GCR(cl_r, k_r)$ and sometimes also overlaps with $GCR(cl, k)$ and R_w. We will show that these overlaps do not reveal a more precise user location to the LSP, i.e., the user's location privacy is protected.

The LSP does not know about the boundary of $GCR(cl_r, k_r)$, which means that the LSP cannot compute $GCR(cl_r, k_r) \cap R_{w+1}$. Thus, the LSP cannot refine a user's location at the time of requesting R_{w+1} or the user's trajectory path from $GCR(cl_r, k_r) \cap R_{w+1}$.

Since the LSP knows $GCR(cl, k)$ and R_w, it can compute the overlaps, $GCR(cl, k) \cap R_{w+1}$ and $R_w \cap R_{w+1}$, when it receives R_{w+1}. However, the availability of $GCR(cl_r, k_r)$ to the user and the option of having different values for δ prevent the LSP to determine whether the user is located within $GCR(cl, k) \cap R_{w+1}$ and $R_w \cap R_{w+1}$ at the time of requesting R_{w+1} or whether the user's trajectory passes through these overlaps.

In summary there is no additional information to render a more precise user position or user trajectory within the rectangle. Thus, every obfuscation rectangle computed using the algorithm REQUEST_PMkNN satisfies the two required conditions (see Definition 2.3) for protecting a user's location privacy. □

6.1.1. The maximum movement bound attack

If a user's maximum velocity is known, then the maximum movement bound attack can render a more precise user position from the overlap of the current obfuscation rectangle and the maximum movement bound of the previous obfuscation rectangle. To prevent this attack, the most recent solution [5] has proposed that R_{w+1} needs to be completely included within the maximum movement bound of R_w, denoted as M_w. As a result, at the time of requesting R_{w+1}, although the user is located in R_{w+1}, the LSP cannot refine the user's more precise position within R_{w+1} as there is no overlap between R_{w+1} and M_w. Our proposed algorithm to generate R_{w+1} can also consider this constraint of M_w whenever the LSP knows the user's maximum velocity. Incorporating the constraint of M_w in our algorithm does not cause any new privacy violation for users.

7. Server-side Processing

For a PMkNN query with a customizable confidence level, an LSP provides the k NNs with the specified confidence level for all points of every requested obfuscation rectangle. Evaluating the k NNs with a specified confidence level for every point of an obfuscation rectangle separately is an expensive operation and doing it continuously for a PMkNN query incurs large overhead. We develop an efficient algorithm that finds the k NNs for every point of an obfuscation rectangle with a specified confidence level in a single search using an R-tree. If the specified confidence level is 1, our algorithm returns exact kNN answers and if the specified confidence level is lower than 1 then the algorithm returns approximate kNN answers. Our proposed algorithm allows an LSP to provide the user with a known region, which helps protecting the user's location privacy from overlapping rectangle attack and further to reduce the overall PMkNN query processing overhead.

We show different properties of a confidence level for an obfuscation rectangle, which we use to improve the efficiency of our algorithms. Let R_w be a user's obfuscation rectangle with center o and corner points $\{c_1, c_2, c_3, c_4\}$, and m_{ij} be the middle point of $\overline{c_i c_j}$, where $(i, j) \in \{(1,2), (2,3), (3,4), (4,1)\}$. To avoid the computation of the confidence level for a data object with respect to every point of R_w while searching for the query answers, we exploit the following properties of the confidence level. We show that if two endpoints, i.e., a corner point and its adjacent middle point or the center and a point in the border of R_w, of a line have a confidence level of at least cl for a data object then every point of the line has a confidence level of at least cl for that data object. Formally, we have the following theorems.

Theorem 7.1. *Let c_i, c_j be any two adjacent corner points of an obfuscation rectangle R_w and m_{ij} be the middle point of $\overline{c_i c_j}$. For $t \in \{i, j\}$, if c_t and m_{ij} have a confidence level of at least cl for a data object p_h then all points in $\overline{m_{ij} c_t}$ have a confidence level of at least cl for p_h.*

Theorem 7.2. *Let o be the center of an obfuscation rectangle R_w, c_i, c_j be any two adjacent corner points of R_w, and c be a point in $\overline{c_i c_j}$. If o and c have a confidence level of at least cl for a data object p_h then all points in $\overline{oc}$ have a confidence level of at least cl for p_h.*

Next we discuss the proof of Theorem 7.1. We omit the proof of Theorem 7.2, since a similar proof technique used for Theorem 7.1 can be applied for Theorem 7.2 by considering o as m_{ij} and c as c_t.

As mentioned in Section 5, our algorithm to evaluate kNN answers expands the known region $C(o, r)$ until the k NNs with the specified confidence level for every point of R_w are found. Since any point outside $C(o, r)$ has a confidence level 0 (see Definition 5.1), $C(o, r)$ needs to be at least expanded until R is within $C(o, r)$ to ensure kNN answers with a specified confidence level greater than 0. Hence, we assume that $R \subset C(o, r)$ at the current state of the search. Let the extended lines $\overrightarrow{om_{ij}}$[4] and $\overrightarrow{oc_t}$ intersect the border of $C(o, r)$ at m'_{ij} and c'_t, respectively, where $t \in \{i, j\}$. Figure 10(a) shows an

[4]We use the symbol $\rightarrow$ for directional line segments.

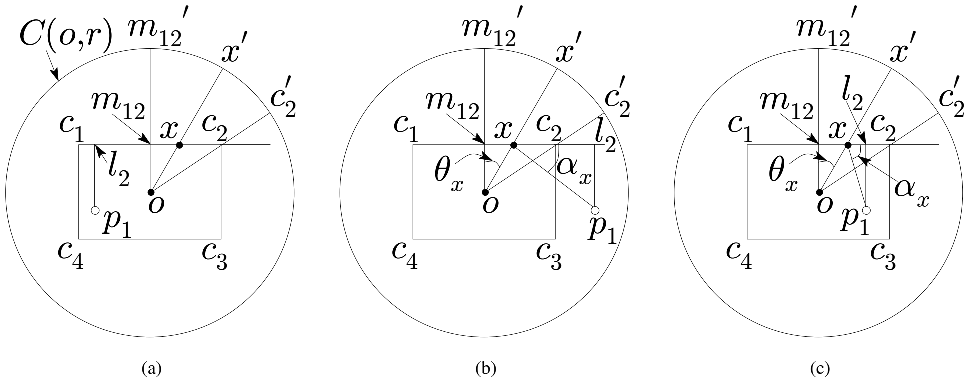


Figure 10: Impact of different positions of a data object

example for $i = 1$, $j = 2$, and $t = j$. For a data object p_h in $C(o,r)$, the confidence levels of c_t and m_{ij}, $CL(c_t, p_h)$ and $CL(m_{ij}, p_h)$, can be expressed as $\frac{dist(c_t,c_t')}{dist(c_t,p_h)}$ and $\frac{dist(m_{ij},m_{ij}')}{dist(m_{ij},p_h)}$, respectively.

Let x be a point in $\overline{m_{ij}c_t}$, and $\overrightarrow{ox}$ intersect the border of $C(o,r)$ at x'. For a data object p_h in $C(o,r)$, the confidence level of x, $CL(x, p_h)$, is measured as $\frac{dist(x,x')}{dist(x,p_h)}$. As x moves from c_t towards m_{ij}, although $dist(x, x')$ always increases, $dist(x, p_h)$ can increase or decrease (does not maintain a single trend) since it depends on the position of p_h within $C(o,r)$. Without loss of generality we consider an example in Figure 10, where p_1 is a data object within $C(o,r)$. Based on the position of p_1 with respect to m_{12} and c_2, we have three cases: the perpendicular from p_1 intersects the extended line $\overrightarrow{c_2m_{12}}$ (see Figure 10(a)) or the extended line $\overrightarrow{m_{12}c_2}$ (see Figure 10(b)) or the segment $\overline{m_{12}c_2}$ (see Figure 10(c)) at l_2. In the first case, $dist(x, p_1)$ decreases as x moves from c_2 towards m_{12} as shown in Figure 10(a). In the second case, $dist(x, p_1)$ decreases as x moves from m_{12} towards c_2 as shown in Figure 10(b). In the third case, $dist(x, p_h)$ is the minimum at $x = l_2$, i.e., $dist(x, p_1)$ decreases as x moves from c_2 or m_{12} towards l_2 as shown in Figure 10(c). From these three cases we observe that for different positions of p_h, $dist(x, p_h)$ can decrease for moving x in both directions, i.e., from c_t towards m_{ij} or from m_{ij} towards c_t.

For the scenario, where $dist(x, p_h)$ decreases as x moves from c_t towards m_{ij} (first case) or from c_t towards l_t (third case), i.e., $dist(x, p_h) \leq dist(c_t, p_h)$, we have the following lemma.

Lemma 7.1. *If $dist(x, p_h) \leq dist(c_t, p_h)$ and $CL(c_t, p_h) \geq cl$ then $CL(x, p_h) \geq cl$, for any point $x \in \overline{m_{ij}c_t}$.*

The proof of this lemma directly follows from $dist(x, x') \geq dist(c_t, c_t')$.

In the other scenario, $dist(x, p_h)$ decreases as x moves from m_{ij} towards c_t (second case) or from m_{ij} towards l_t (third case). In the general case, let u_t be a point that represents c_t for the second case and l_t for the third case. To prove that $CL(x, p_h) \geq cl$, in contrast to Lemma 7.1 where we only need to have $CL(c_t, p_h) \geq cl$, for the current scenario we need to have the confidence level at least equal to cl for p_h at both end points, i.e., m_{ij} and u_t. According to the given conditions of Theorem 7.1, we already have $CL(m_{ij}, p_h) \geq cl$ and $CL(c_t, p_h) \geq cl$. Since u_t is c_t in the second case and l_t in the third case, we need to compute the confidence level of l_t for p_h in the third case and using Lemma 7.1 we find that $CL(l_t, p_h) \geq cl$. Thus, we have the confidence level of both m_{ij} and u_t for p_h at least equal to cl.

However, showing $CL(x, p_h) \geq cl$ if both m_{ij} and u_t have a confidence level of at least cl for p_h is not straightforward, because in the current scenario both $dist(x, p_h)$ and $dist(x, x')$ decrease with the increase of $dist(m_{ij}, x)$. Thus, we need to compare the rate of decrease for $dist(x, x')$ and $dist(x, p_h)$ as x moves from m_{ij} to u_t. Assume that $\angle xom_{ij} = \theta_x$ and $\angle p_h x l_t = \alpha_x$. The range of θ_x can vary from 0 to θ, where $\theta_{m_{ij}} = 0$, $\theta_{u_t} = \theta$, and $\theta \leq \frac{\Pi}{4}$. For a fixed range of θ_x the range of α_x, $[\alpha_{m_{ij}}, \alpha_{u_t}]$, can have any range from $[0, \frac{\Pi}{2}]$ depending upon the position of p_h. We express $dist(x, x')$ and $dist(x, p_h)$ as follows:

$$dist(x, x') = r - dist(o, m_{ij}) \times \sec\theta_x$$

$$dist(x, p_h) = \begin{cases} dist(p_h, l_t) \times \csc\alpha_x & \text{if } \alpha_x \neq 0 \\ dist(m_{ij}, p_h) - dist(m_{ij}, x) & \text{otherwise.} \end{cases}$$

The rate of decrease for $dist(x, x')$ and $dist(x, p_h)$ are not comparable by computing their first order deriva-

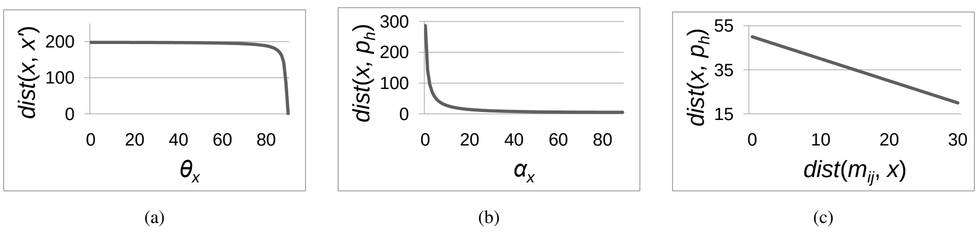


Figure 11: Curve sketching

tive as they are expressed with different variables and there is no fixed relation between the range of θ_x and α_x. Therefore, we perform a curve sketching and consider the second order derivative in Figure 11. From the second order derivative, we observe in Figure 11(a) that the rate of decreasing rate of $dist(x, x')$ increases with the increase of θ_x, whereas in Figure 11(b) the rate of decreasing rate of $dist(x, p_h)$ decreases with the increase of α_x for $\alpha_x \neq 0$ and in Figure 11(c) the rate of decreasing rate remains constant with the increase of $dist(m_{ij}, x)$ for $\alpha_x = 0$. The different trends of the decreasing rate and the constraint of confidence levels at two end points m_{ij} and u_t allow us to make a qualitative comparison between the rate of decrease for $dist(x, x')$ and $dist(x, p_h)$ with respect to the common metric $dist(m_{ij}, x)$, as $dist(m_{ij}, x)$ increases with the increase of both θ_x and α_x for a fixed p_h. We have the following lemma.

Lemma 7.2. *Let $dist(x, p_h)$ decrease as x moves from m_{ij} to u_t for any point $x \in \overline{m_{ij}u_t}$. If $CL(m_{ij}, p_h) \geq cl$ and $CL(u_t, p_h) \geq cl$, then $CL(x, p_h) \geq cl$.*

Proof. (By contradiction) Assume to the contrary that there is a point $x \in \overline{m_{ij}u_t}$ such that $CL(x, p_h) < cl$, i.e., $\frac{dist(x,x')}{dist(x,p_h)} < cl$. Then we have the following relations.

$$\frac{dist(m_{ij}, m'_{ij}) - dist(x, x')}{dist(m_{ij}, x)} > \frac{dist(m_{ij}, p_h) - dist(x, p_h)}{dist(m_{ij}, x)} \quad (1)$$

$$\frac{dist(x, x') - dist(u_t, u'_t)}{dist(x, u_t)} < \frac{dist(x, p_h) - dist(u_t, p_h)}{dist(x, u_t)} \quad (2)$$

Since we know that for $dist(x, x')$, the rate of decreasing rate increases with the increase of $dist(m_{ij}, x)$ and for $dist(x, p_h)$, the rate of decreasing rate decreases or remains constant with the increase of $dist(m_{ij}, x)$, we have the following relations.

$$\frac{dist(m_{ij}, m'_{ij}) - dist(x, x')}{dist(m_{ij}, x)} < \frac{dist(x, x') - dist(u_t, u'_t)}{dist(x, u_t)} \quad (3)$$

$$\frac{dist(m_{ij}, p_h) - dist(x, p_h)}{dist(m_{ij}, x)} \geq \frac{dist(x, p_h) - dist(u_t, p_h)}{dist(x, u_t)} \quad (4)$$

From Equations 1, 2, and 3 we have,

$$\frac{dist(m_{ij}, p_h) - dist(x, p_h)}{dist(m_{ij}, x)} < \frac{dist(x, p_h) - dist(u_t, p_h)}{dist(x, u_t)}$$

which contradicts Equation 4, i.e., our assumption. □

Finally, from Lemmas 7.1 and 7.2, we can conclude that if $CL(c_t, p_h) \geq cl$ and $CL(m_{ij}, p_h) \geq cl$, then $CL(x, p_h) \geq cl$ for any point $x \in \overline{m_{ij}c_t}$, which proves Theorem 7.1.

7.1. Algorithms

We develop an efficient algorithm, Clappinq (Confidence Level Aware Privacy Protection In Nearest Neighbor Queries), that finds the k NNs for an obfuscation rectangle with a specified confidence level using an R-tree (see Algorithm 3).

As mentioned in Section 5, the basic idea of our algorithm is to start a best first search (BFS) considering the center o of the given obfuscation rectangle R_w as the query point and continue the search until the k NNs with a confidence level of at least cl are found for all points of R_w. The known region $C(o, r)$ is the search region covered by BFS and P is the set of data objects located within $C(o, r)$. Q_p is a priority queue used to maintain the ordered data objects and R-tree nodes based on the minimum distance between the query point o and the data objects/MBRs of R-tree nodes (by using the function *MinDist*). Since the size of the candidate answer set is unknown, we use *status* to control the execution of the BFS. Based on the values of *status*, the BFS can have three states: (i) when $status = 0$, each time the BFS discovers the next nearest data object, it checks whether *status* needs to be updated, (ii) when $status > 0$, the BFS executes until the radius of the known region becomes greater than the value of *status*, and (iii) when

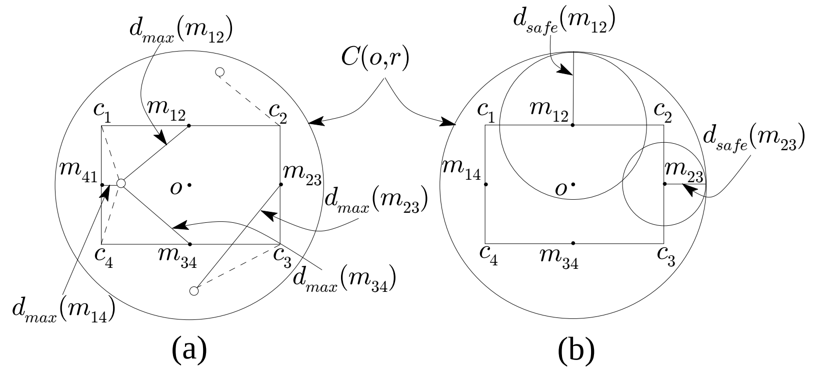


Figure 12: (a) $d_{max} = d_{max}(m_{23})$ and (b) $d_{safe} = d_{safe}(m_{23})$

Algorithm 3: CLAPPINQ(R_w, cl, k)

```
3.1  $P \leftarrow \emptyset$
3.2  $status \leftarrow 0$
3.3  $Enqueue(Q_p, root, 0)$
3.4  while $Q_p$ is not empty and $status \geq 0$ do
3.5      $p \leftarrow Dequeue(Q_p)$
3.6      $r \leftarrow MinDist(o, p)$
3.7      if $status > 0$ and $status < r$ then
3.8          $status \leftarrow -1$
3.9      if $p$ is a data object then
3.10         $P \leftarrow P \cup p$
3.11         if $status = 0$ then
3.12             $status \leftarrow$
                 $UpdateStatus(R_w, cl, k, P, r)$
3.13     else
3.14         for each child node $p_c$ of $p$ do
3.15             $d_{min}(p_c) \leftarrow MinDist(o, p_c)$
3.16             $Enqueue(Q_p, p_c, d_{min}(p_c))$
3.17 return $P$;
```

$status = -1$, the BFS terminates. Initially, $status$ is set to 0. Each time a data object/R-tree node p is dequeued from Q_p the current radius r is updated. When p represents a data object, then p is added to the current candidate set P and the procedure $UpdateStatus$ is called if $status$ equals 0.

The pseudo code for $UpdateStatus$ is shown in Algorithm 4. The notations used for this algorithm are summarized below.

- $count(c_t, cl, P)$: the number of data objects in P for which a corner point c_t of R_w has a confidence level of at least cl.
- d_i^k (d_j^k): the kth minimum distance from a middle point m_{ij} of R_w to the data objects in P_i (P_j), where P_i (P_j) $\subset P$ and P_i (P_j) is the set of data objects with respect to c_i (c_j) with a confidence level of at least cl.
- d_{max}: the maximum of all $d_{max}(m_{ij})$, where each $d_{max}(m_{ij})$ is the maximum of d_i^k and d_j^k (see Figure 12(a)).
- d_{safe}: the minimum distance of all $d_{safe}(m_{ij})$, where $d_{safe}(m_{ij})$ represents the radius of the maximum circular region within $C(o, r)$ centered at m_{ij} (see Figure 12(b)).

$UpdateStatus$ first updates $count(c_t, cl, P)$ using the function $UpdateCount$. For each $p \in P$, $UpdateCount$ increments $count(c_t, cl, P)$ by one if $CL(c_t, p) >= cl$. Note that corner points of R_w can have more than k data objects with a confidence level of at least cl because the increase of r for a corner point of R_w can make other corner points to have more than k data objects with a confidence level of at least cl. In the next step if $count(c_t, cl, P)$ is less than k for any corner point c_t of R_w, $UpdateStatus$ returns the control to Algorithm 3 without changing $status$. Otherwise, it computes the radius of the required known region for ensuring the k NNs with respect to R_w and cl (Lines 3.5-3.16). For each m_{ij}, $UpdateStatus$ first computes d_i^k and d_j^k with the function K_{min} and takes the maximum of d_i^k and d_j^k as $d_{max}(m_{ij})$. Then $UpdateStatus$ finds d_{max} (Lines 3.10-3.11) and d_{safe} (Line 3.12). Finally, $UpdateStatus$ checks if the size of the current $C(o, r)$ is already equal or greater than the required size. If this is the case then the algorithm returns $status$ as -1, otherwise the value of the radius for the required known region. After the call of $UpdateStatus$, CLAPPINQ continues the BFS if $status \geq 0$ and terminates if $status = -1$. For $status$ greater than 0, each time a next nearest data object/MBR is found, CLAPPINQ updates *status* to -1 if r becomes greater than *status* (Lines 3.7-3.8).

```
Algorithm 4: UpdateStatus($R_w, cl, k, P, r$)
4.1  $UpdateCount(R, cl, k, P, r, count)$
4.2  if $count(c_t, cl, P) \neq k$, for any corner point
     $c_t \in R$ then
4.3      return 0
4.4  else
4.5      $d_{max} \leftarrow 0$
4.6      for each middle point $m_{i,j}$ do
4.7          $d_i^k \leftarrow K_{min}(m_{ij}, c_i, cl, k, P)$
4.8          $d_j^k \leftarrow K_{min}(m_{ij}, c_j, cl, k, P)$
4.9          $d_{max}(m_{ij}) \leftarrow \max\{d_i^k, d_j^k\}$
4.10         if $d_{max}(m_{ij}) > d_{max}$ then
4.11             $d_{max} \leftarrow d_{max}(m_{ij})$
4.12     $d_{safe} \leftarrow r - \frac{1}{2} \times \max\{||\overline{c_1c_2}||, ||\overline{c_2c_3}||\}$
4.13     if $cl \times d_{max} > d_{safe}$ then
4.14         return $(r + cl \times d_{max} - d_{safe})$
4.15     else
4.16         return $-1$
```

In summary, CLAPPINQ works in three steps. In step 1, it runs the BFS from o until it finds the k NNs with a confidence level of at least cl for all corner points of R_w. In step 2, from the current set of data objects it computes the radius of the required known region to confirm that the answer set includes the k NNs with a confidence level of at least cl with respect to all points of R_w. Finally, in step 3, it continues to run the BFS until the radius of the current known region is equal to the required size.

Figure 13 shows an example of the execution of CLAPPINQ for $k = 1$ and $cl = 1$. Data objects are labeled in order of the increasing distance from o. CLAPPINQ starts its search from o and continues until the NNs with respect to four corner points are found as shown in Figure 13(a). The circles with ash border show the continuous expanding of the known region and the circle with black border represents the current known region. The data objects p_4, p_7, p_5, and p_3 are the NNs with $cl = 1$ from c_1, c_2, c_3, and c_4, respectively because the four circles with a dashed border are completely within the known region. In the next step, the algorithm finds the maximum of d_i^k and d_j^k for each m_{ij}. The distances d_2^1 (=$dist(m_{12}, p_7)$), d_2^1 (=$dist(m_{12}, p_7)$) (or d_3^1 (=$dist(m_{23}, p_5)$)), d_4^1 (=$dist(m_{34}, p_3)$), d_1^1 (=$dist(m_{41}, p_4)$) are the maximum with respect to m_{12}, m_{23}, m_{34}, and m_{41}, respectively. Finally, CLAPPINQ expands the search so that the four circles with dashed border centered at m_{12}, m_{23}, m_{34}, and m_{41} and having radius d_2^1, d_2^1 (or d_3^1), d_4^1, and d_1^1, respectively, are included in the known region (see Figure 13(b)). Therefore, the search stops when p_9 is discovered and P includes p_1 to p_9.

The following theorem shows the correctness for CLAPPINQ.

Theorem 7.3. CLAPPINQ *returns P, a candidate set of data objects that includes the k NNs with a confidence level of at least cl for every point of the obfuscation rectangle R_w.*

PROOF. CLAPPINQ expands the known region $C(o, r)$ from the center o of the obfuscation rectangle R_w until it finds the k NNs with a confidence level of at least cl for all corner points of R_w. Then it extends $C(o, r)$ to ensure that the confidence level of each middle point m_{ij} is at least cl for both sets of k nearest data objects for which c_i and c_j have a confidence level of at least cl. According to Theorem 7.1, this ensures that any point in $\overline{m_{ij}c_i}$ or $\overline{m_{ij}c_j}$ has a confidence level of at least cl for k data objects. Again from Lemma 6.1, we know that if a point has k data objects with a confidence level of at least cl then it also has a confidence level of at least cl for its k NNs. Thus, P contains the k NNs with a confidence level of at least cl for all points of the border of R_w.

To complete the proof, next we need to show that P also contains the k nearest data objects with a confidence level of at least cl for all points inside R_w. The confidence level of the center o of R_w for a data object p_h within the known region $C(o, r)$ is always 1 because $C(o, r)$ is expanded from o and we have $dist(o, p_h) \leq r$. Since we have already shown that P includes the k NNs with a confidence level of at least cl for all points of the border of R_w, according to Theorem 7.1 and Lemma 6.1, P also includes the k NNs with a confidence level of at least cl for all points inside R_w. □

We have proposed the fundamental algorithm and there are many possible optimizations of it. For example, one optimization could merge overlapping obfuscation rectangles requested by different users at the same time, which will also avoid redundant computation. Another optimization could exploit that R_w and R_{w+1} may have many overlapping NNs. However, the focus of this paper is protecting location privacy of users while answering MkNN queries, and exploring all possible optimizations of the algorithm is beyond the scope of this paper.

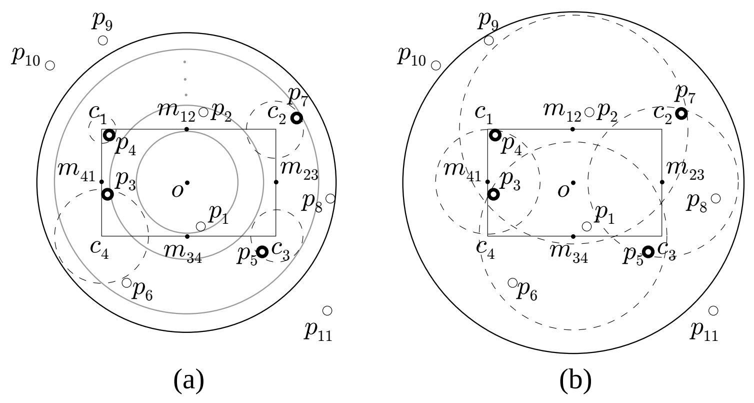


Figure 13: Steps of CLAPPINQ: an example for $k = 1$ and $cl = 1$

8. Experiments

In this section, we present an extensive experimental evaluation of our proposed approach. In our experiments, we use both synthetic and real data sets. Our two synthetic data sets are generated from uniform (U) and Zipfian (Z) distribution, respectively. The synthetic data sets contain locations of 20,000 data objects and the real data set contains 62,556 postal addresses from California. These data objects are indexed using an R^*-tree [46] on a server (the LSP). We run all of our experiments on a desktop with a Pentium 2.40 GHz CPU and 2 GByte RAM.

In Section 8.1, we evaluate the efficiency of our proposed algorithm, CLAPPINQ, to find k NNs with a specified confidence level for an obfuscation rectangle. We measure the query evaluation time, I/Os, and the candidate answer set size as the performance metric. In Section 8.2, we evaluate the effectiveness of our technique for preserving location privacy for MkNN queries.

8.1. *Efficiency of* CLAPPINQ

There is no existing algorithm to process a PMkNN query. An essential component of our approach for a PMkNN query is an algorithm for the LSP to evaluate a kNN query with respect to an obfuscation rectangle. In this set of experiments we compare our proposed kNN algorithm, CLAPPINQ, with Casper [14], because Casper is the only existing related algorithm that can be adapted to process a PMkNN query; further, even if we adapt it can only support $k = 1$. To be more specific, our privacy aware approach for MkNN queries needs an algorithm that returns the *known region* in addition to the set of k NNs with respect to an obfuscation rectangle. Among all existing kNN algorithms [12, 14, 21, 42, 43, 44] only Casper supports the known region and if Casper were as efficient as CLAPPINQ, then in theory we could extend Casper using our privacy protection approach to process PMkNN queries for the restricted case $k = 1$.

We set the data space as 10,000 × 10,000 square units. For each set of experiments in this section, we generate 1000 random obfuscation rectangles of a specified area, which are uniformly distributed in the total data space. We evaluate a kNN query with respect to 1000 obfuscation rectangles and measure the average performance with respect to a single obfuscation rectangle for Casper and CLAPPINQ in terms of the query evaluation time, the number of page accesses, i.e., I/Os, and the candidate answer set size. The page size is set to 1 KB which corresponds to a node capacity of 50 entries.

Note that, in our experiments, the communication amount (i.e., the answer set size) represents the communication cost independent of communication link (e.g., wireless LANs, cellular link) used. The communication delay can be approximated from the known latency of the communication link. In our technique, sometimes the answer set size may become large to satisfy the user's privacy requirement. Though the large answer set size may result in a communication delay, nowadays this should not be a problem. The latency of wireless links has been significantly reduced, for example HSPA+ offers a latency as low as 10ms. Furthermore, our analysis represents the communication delay scenario in the worst case. In practice, the latency of first packet is higher than the subsequent packets and thus, the communication delay does not increase linearly with the increase of the answer set size.

In different sets of experiments, we vary the following parameters: the area of the obfuscation rectangle,

Parameter	Range	Default
Obfuscation rectangle area	0.001% to 0.01%	0.005%
Obfuscation rectangle ratio	1, 2, 4, 8	1
Specified confidence level *cl*	0.5 to 1	1
Specified number of NNs *k*	1 to 20	1
Synthetic data set size	5K, 10K, 15K, 20K	20K

Table 2: Experimental Setup

the ratio of the length and width of the obfuscation rectangle, the specified confidence level, the specified number of NNs and the synthetic data set size. Table 2 shows the range and default value for each of these parameters. We set 0.005% of the total data space as the default area for the obfuscation rectangle, since it reflects a small suburb in California (about 20 km^2 with respect to the total area of California) and is sufficient to protect privacy of a user's location. The thinner an obfuscation rectangle, the higher the probability to identify a user's trajectory [47]. Hence, we set 1 as a default value for the ratio of the obfuscation rectangle to ensure the privacy of the user. The original approach of Casper does not have the concept of confidence level and only addresses 1NN queries. To compare our approach with Casper, we set the default value in CLAPPINQ for k and the confidence level as 1.

In Sections 8.1.1 and 8.1.2, we evaluate and compare CLAPPINQ with Casper. In Section 8.1.3, we study the impact of k and the confidence level only for CLAPPINQ as Casper cannot be directly applied for $k > 1$ and has no concept of a confidence level.

8.1.1. The effect of obfuscation rectangle area

In this set of experiments, we vary the area of obfuscation rectangle from 0.001% to 0.01% of the total data space. A larger obfuscation rectangle represents a more imprecise location of the user and thus ensures a higher level of privacy. We also vary the obfuscation rectangle ratio as 1,2,4, and 8. A smaller ratio of the width and length of the obfuscation rectangle provides the user with a higher level of privacy.

Figures 14(a) and 14(b) show that CLAPPINQ is on an average 3times faster than Casper for all data sets. The I/Os are also at least 3 times less than Casper (Figures 14(c) and 14(d)). The difference between the answer set size for CLAPPINQ and Casper is not prominent. However, in most of the cases CLAPPINQ results in a smaller answer set compared with that of Casper (Figures 14(e) and 14(f)). We also observe that the performance is better when the obfuscation rectangle is a square and it continues to degrade for a larger ratio in both CLAPPINQ and Casper (Figures 14(b), 14(d), and 14(f)).

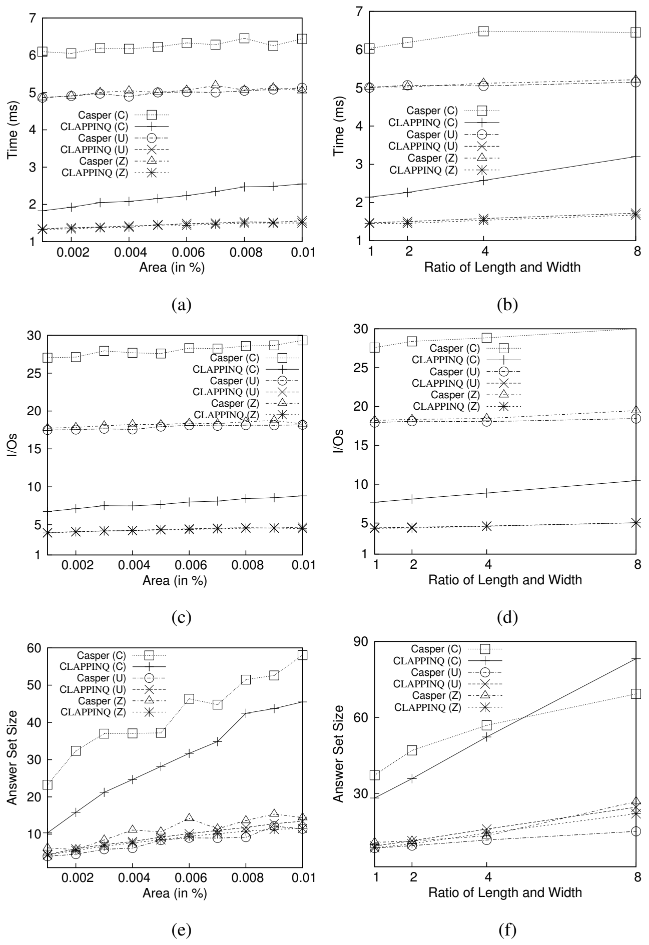
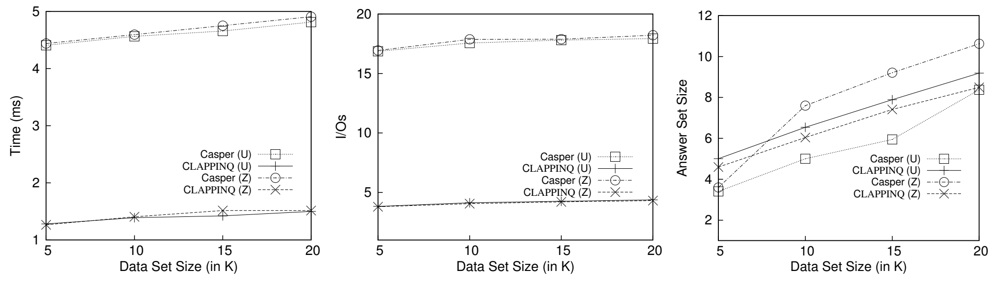


Figure 14: The effect of obfuscation rectangle area and ratio

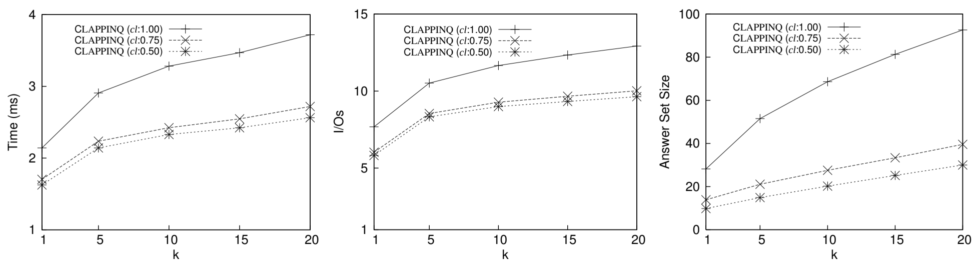
8.1.2. The effect of the data set size

We vary the size of the synthetic data set as 5K, 10K, 15K and 20K, and observe that CLAPPINQ is significantly faster than that of Casper for any size of data set. Figure 15 shows the results for the query evaluation



(a) (b) (c)

Figure 15: The effect of data set size



(a) (b) (b)

Figure 16: The effect of the parameter k and confidence level

time, I/Os and the answer set size. We find that CLAPPINQ is at least 3 times faster and the I/Os of CLAPPINQ is at least 4 times less than that of Casper. The time, the I/Os and the answer set size slowly increases with the increase of data set size.

8.1.3. *The effect of k and the confidence level*

In this set of experiments, we observe that the query evaluation time, I/Os, and the answer set size for CLAPPINQ increase with the increase of k for all data sets. However, these increasing rates decrease as k increases (Figure 16 for the California data set). We also vary the confidence level cl and expect that a lower cl incurs less query processing and communication overhead. Figure 16 also shows that the average performance improves as cl decreases and the improvement is more pronounced for higher values of cl. For example, the answer set size reduces by an average factor of 2.35 and 1.37 when cl decreases from 1.00 to 0.75 and from 0.75 to 0.50, respectively.

8.1.4. CLAPPINQ *vs. Casper for PMkNN queries*

The paper that proposed Casper [14] did not address location privacy for M*k*NN queries. Even if we extend it for PM*k*NN queries using our technique, Casper would only work for $k = 1$. More importantly, since we have found that CLAPPINQ is at least 2 times faster and requires at least 3 times less I/Os than Casper for finding the NNs for an obfuscation rectangle, and an M*k*NN query requires the evaluation of a large number of consecutive obfuscation rectangles, CLAPPINQ would outperform Casper by a greater margin for PM*k*NN queries. Therefore, we do not perform such experiments and conclude that CLAPPINQ is efficient than Casper for PM*k*NN queries.

8.2. *Effectiveness of our privacy protection technique*

We develop the first approach to protect a user's location privacy from the overlapping rectangle attack and the combined attack that also integrates the maximum movement bound attack if the user's maximum velocity is known to the LSP. Thus, our technique does not allow the LSP to refine the user locations within the obfuscation rectangles by applying these attacks. Since there is no other approach that addresses the overlapping rectangle attack, we cannot compare our technique with others in the experiments. Even the naïve technique that we discussed in Section 2 does not satisfy the definition of an M*k*NN query. To avoid the overlapping rectangle attack, the naïve technique sends disjoint rectangles to

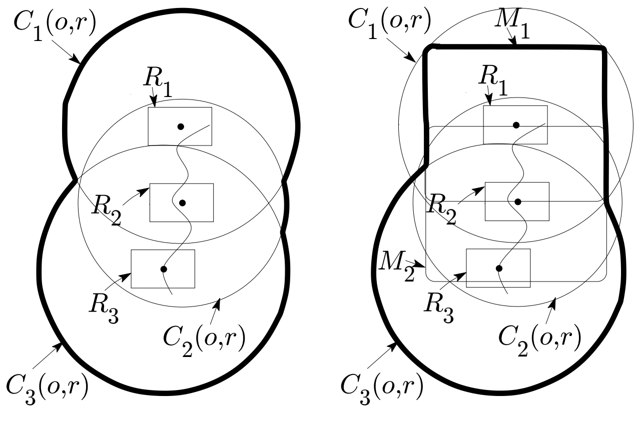


Figure 17: The bold line shows the trajectory area if the maximum velocity is (a) unknown to the LSP, (b) known to the LSP

the LSP and thus cannot ensure *k*NN answers for every point of a user's trajectory, which is a necessary condition for an M*k*NN query. In our experiments, we evaluate the effectiveness of our privacy protection technique by varying different parameters.

The user reveals a sequence of obfuscation rectangles to the LSP while accessing a PM*k*NN query. The LSP cannot refine a user location within an individual obfuscation rectangle by applying the overlapping rectangle attack. In addition, we have assumed that the LSP does not have background knowledge about the user's location (see Section 2.2 for details) as the user's location is considered private and sensitive data. Note that since the LSP does not know distribution of any parameters such as confidence level or k, any probabilistic attack is also not applicable to our case. In our experiments, we define two measures for location privacy based on what information a user reveals about location: (i) the trajectory area, i.e., the approximated location of the user's trajectory by the LSP, and (ii) the frequency, i.e., the number of requested obfuscation rectangles per a user's trajectory. The larger the trajectory area, the higher the privacy for the user. This is because the probability is high for a large trajectory area to contain different sensitive locations and the probability is low that an LSP can link the user's trajectory with a specific location. On the other hand, a lower frequency for a fixed rectangle area represents high level of location privacy since the LSP has less information about the user's locations.

The trajectory area is computed from the available knowledge of the LSP. The LSP knows the obfuscated rectangles provided by a user and the known region for each obfuscated rectangle. The LSP does not know the user's required confidence level cl_r and the required number of NNs k_r and thus, cannot compute $GCR(cl_r, k_r)$. Although the LSP can compute $GCR(cl, k)$ from the user's specified confidence level cl and the specified number of NNs k, $GCR(cl, k)$ does not guarantee that the user's location resides in $GCR(cl, k)$ for the current obfuscation rectangle. We know that the user needs to reside within $GCR(cl_r, k_r)$ of the current obfuscation rectangle to ensure the required confidence level for the required number of NNs. However, the LSP knows the known region $C(o, r)$ and that $GCR(cl_r, k_r)$ must be inside the known region of the current obfuscation rectangle because the confidence level of the user for any data object outside the known region is 0. Thus, the trajectory area for a user's trajectory is defined as the union of the known regions with respect to the set of obfuscation rectangles provided by the user for that trajectory. On the other hand, when the LSP knows the maximum velocity, then the LSP can use the maximum movement bound in addition to the known region to determine the trajectory area.

Definition 8.1. ***(Trajectory Area)*** *Let* $\{R_1, R_2, ..., R_n\}$ *be a set of n consecutive rectangles requested by a user to an LSP in an MkNN query,* $C_i(o, r)$ *be the known region corresponding to* R_i*, and* M_i *be the maximum movement bound corresponding to* R_i*. The trajectory area is computed as* $\cup_{1 \le i \le n-1}(C_i(o, r) \cap M_i) \cup C_n(o, r)$*. If the maximum bound is unknown to the LSP then the trajectory area is expressed as* $\cup_{1 \le i \le n} C_i(o, r)$*.*

Figures 17(a) and 17(b) show trajectory areas when the maximum velocity is either unknown or known to the LSP, respectively. In our experiments, we compute the trajectory area through Monte Carlo Simulation. We randomly generate 1 million points in the total space. For the overlapping rectangle attack, we determine the trajectory area as the percentage of points that fall inside

$\cup_{1\leq i\leq n} C_i(o,r)$. On the other hand, for the combined attack (i.e., the maximum velocity is known to the LSP), we determine the trajectory area as the percentage of points that fall inside $\cup_{1\leq i\leq n-1}(C_i(o,r) \cap M_i) \cup C_n(o,r)$.

Thus, the trajectory area is measured as percentage of the total data space. On the other hand, the frequency is measured as the number of requested obfuscation rectangles per trajectory of length 5000 units in the data space of 10,000 × 10,000 square units. Note that for the sake of simplicity, in our experiments, we use fixed area for every obfuscation rectangle requested for a trajectory.

To simulate moving users, we first randomly generate starting points of 20 moving users that are uniformly distributed in the data space and then generate the complete movement path (trajectory) for each of these starting points. Each trajectory has a length of 5000 units and consists of a series of random points; consecutive points are connected with segments of a random length between 1 to 10 units. We generate an obfuscation rectangle of a specified area for user requests. Its default area is 0.005% of the total data space, which reflects a small suburb in California (about 20 km^2 with respect to the total area of California) that is sufficient to protect a user's location. Since the obfuscation rectangle generation is random, we repeat every experiment 25 times for each trajectory and present the average performance results. Although Algorithm 2 supports different cl, cl_r, k, k_r and δ in consecutive obfuscation rectangle requests, we use fixed values for the sake of simplicity. The safe distance δ has a default value of 10. In the combined attack, we set the user's maximum velocity as 60 km/h. For simplicity, we assume that the user also moves at constant velocity of 60 km/h.

The trajectory area is computed based on known regions, where the size of the known region is affected by the distribution of data objects, the obfuscation rectangle area, confidence level and the number of nearest data objects. We use both real and synthetic data sets in our experiment, where two synthetic data sets are generated using uniform and Zipfian distributions of data objects. In the next sections, we present our experimental results showing the effect of obfuscation rectangle area, cl_r, cl, k_r, k and δ on the level of a user's location privacy.

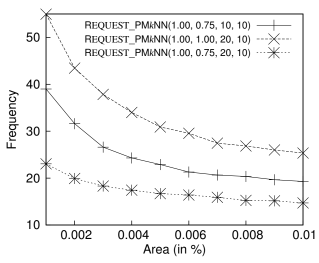
8.2.1. The effect of obfuscation rectangle area

In this set of experiments, we evaluate the effect of obfuscation rectangle area on the three privacy protection options for our algorithm REQUEST_PM*k*NN. In the first option, the user sacrifices the accuracy of answers to achieve location privacy. Using this option, the user's required confidence level is lower than 1 and the user specifies higher confidence level to the LSP than her required one. In this set of experiments, we represent the first option for our algorithm REQUEST_PM*k*NN(cl, cl_r, k, k_r) as REQUEST_PM*k*NN(1,0.75,10,10), where the user hides the required confidence level 0.75 from the LSP, instead specifies 1 for the confidence level. In the second option, the user does not sacrifice the accuracy of the answers for her location privacy; instead the user specifies a higher number of data objects to the LSP than her required one. For the second option, we set the parameters of REQUEST_PM*k*NN(cl, cl_r, k, k_r) as REQUEST_PM*k*NN(1,1,20,10). In the third option, the user hides both of the required confidence level and the required number of data objects. Thus, the third option is represented as REQUEST_PM*k*NN(1,0.75,20,10).

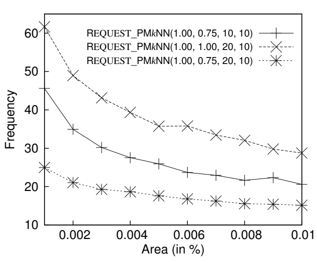
We vary the obfuscation rectangle area from 0.001% to 0.01% of the total data space. For all the three options, we observe in Figures 18(a) and 18(b) that the frequency decreases with the increase of the obfuscation rectangle area for both overlapping rectangle attack and combined attack, respectively. On the other hand, Figures 18(c) and 18(d) show that the trajectory area increases with the increase of the obfuscation rectangle area for overlapping rectangle attack and combined attack, respectively. Thus, the larger the obfuscation rectangle area, the higher the location privacy in terms of both frequency and trajectory area. This is because the larger the obfuscation rectangle the higher the probability that the obfuscation rectangle covers a longer part of a user's trajectory.

Figures 18(a) and 18(b) also show that the frequency for hiding both confidence level and the number of NNs is smaller than those for hiding them independently for any obfuscation rectangle area, since each of them contributes to extend the $GCR(cl_r, k_r)$. In addition, we observe that the rate of decrease of frequency with the increase of the obfuscation rectangle area is more significant for the option of hiding the confidence level than the option of hiding the number of NNs.

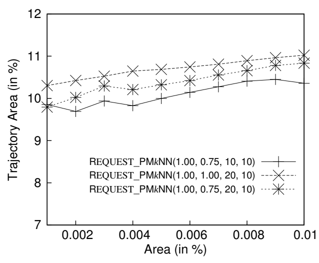
We observe from Figures 18(a) and 18(b) that the frequency in the combined attack is higher than that of the overlapping rectangle attack. The underlying cause is as follows. In our algorithm to protect the overlapping rectangle attack the obfuscation rectangle needs to be generated inside the current known region. On the other hand, in case of the combined attack the obfuscation rectangle needs to be inside the intersection of maximum movement bound and the known region. Due to the stricter constraints while generating the obfuscation rectangle to overcome the combined attack, the frequency becomes higher for the combined attack than



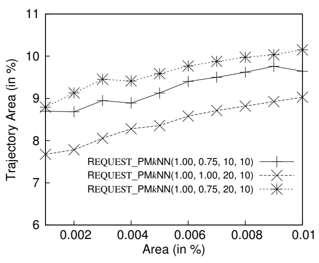
(a) Overlapping Rectangle Attack



(b) Combined Attack

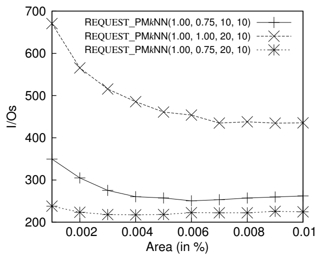


(c) Overlapping Rectangle Attack

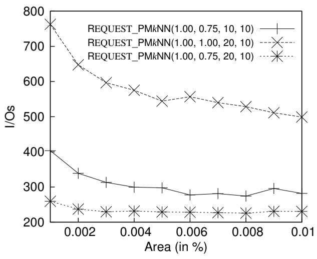


(d) Combined Attack

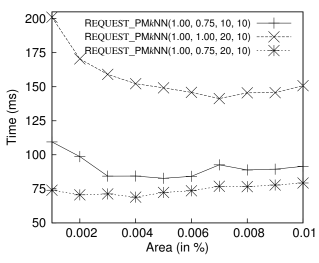
Figure 18: The effect of the obfuscation rectangle area on the level of location privacy for the California data set



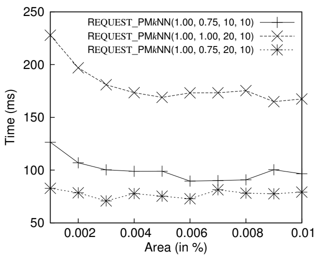
(a) Overlapping Rectangle Attack



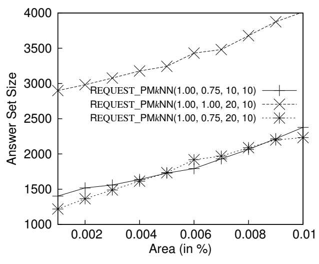
(b) Combined Attack



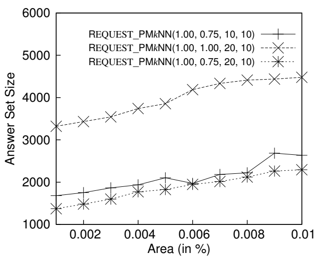
(c) Overlapping Rectangle Attack



(d) Combined Attack



(e) Overlapping Rectangle Attack



(f) Combined Attack

Figure 19: The effect of the obfuscation rectangle area on the query processing performance for the California data set

that of the overlapping rectangle attack. For the same reason, the trajectory area is smaller for the combined attack than that of the overlapping rectangle attack as shown in Figures 18(c) and 18(d).

In Figures 19(a)-(d), we observe that both I/Os and time follow the similar trend of frequency, as expected. On the other hand, the answer set size shows an increasing trend with the increase of the obfuscation rectangle area in Figure 19(e)-(f). We also run all of these experiments for other data sets and the results show similar trends to those of California data set except that of the answer set size. The different trends of the answer set size may result from different density and distributions of data objects.

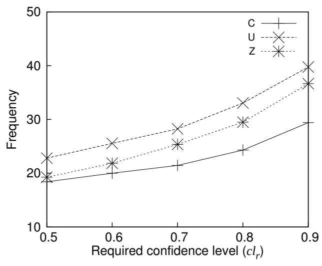
8.2.2. *The effect of cl_r and cl*

In these experiments, we observe the effect of the required and specified confidence level on the level of location privacy. We vary the value of the required confidence level and the specified confidence level from 0.5 to 0.9 and 0.6 to 1, respectively.

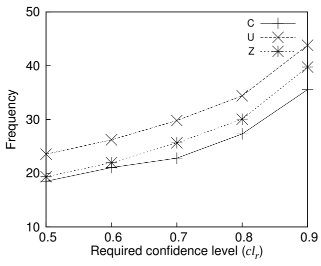
Figures 20(a)-(b) show that the frequency increases with the increase of the required confidence level cl_r for a fixed specified confidence level $cl = 1$. We observe that the larger the difference between required and specified confidence level, the higher the level of location privacy in terms of the frequency because the larger difference causes the larger extension of $GCR(cl_r, k_r)$. On the other hand, Figures 20(c)-(d) show that the trajectory area almost remain constant for different cl_r as cl remains fixed.

Figure 21(a)-(b)) shows that the frequency decreases with the increase of the specified confidence level cl for a fixed required confidence level $cl_r = 0.5$. With the increase of cl, for a fixed cl_r, the extension of $GCR(cl_r, k_r)$ becomes larger and the level of location privacy in terms of frequency increases. On the other hand, Figures 21(c)-(d) show that the trajectory area increases with the increase of cl, as expected.

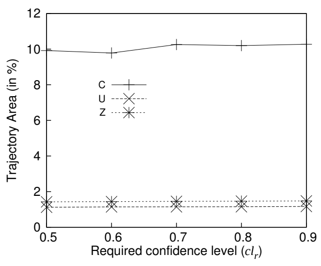
We observe from Figures 20 and 21 that the frequency is higher and the trajectory area is smaller in case of the combined attack than those for the case of the overlapping rectangle attack, which is expected due



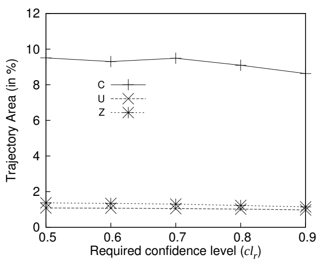
(a) Overlapping Rectangle Attack



(b) Combined Attack

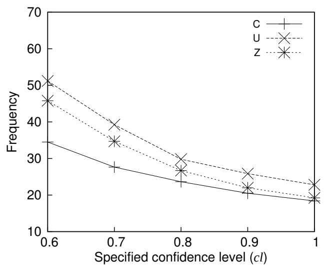


(c) Overlapping Rectangle Attack

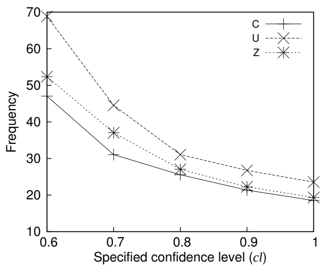


(d) Combined Attack

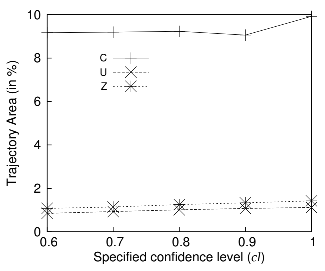
Figure 20: The effect of hiding the required confidence level on the level of location privacy



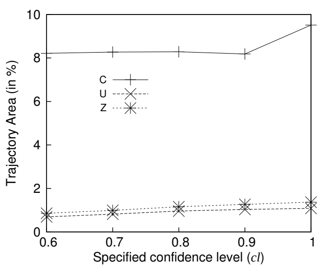
(a) Overlapping Rectangle Attack



(b) Combined Attack

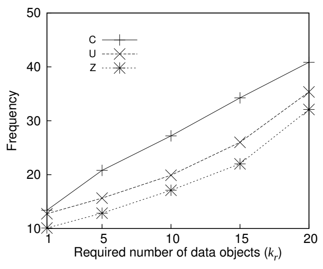


(c) Overlapping Rectangle Attack

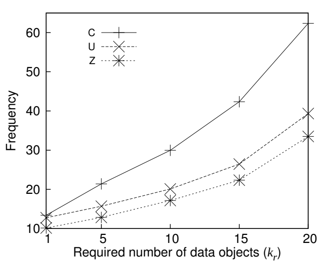


(d) Combined Attack

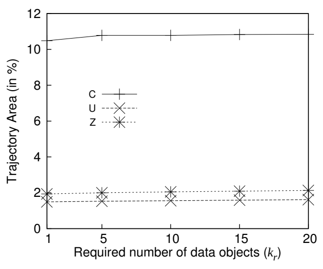
Figure 21: The effect of hiding the specified confidence level on the level of location privacy



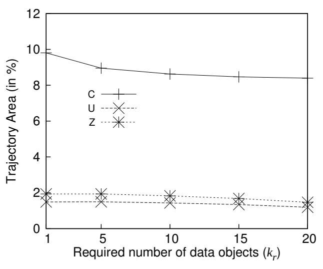
(a) Overlapping Rectangle Attack



(b) Combined Attack

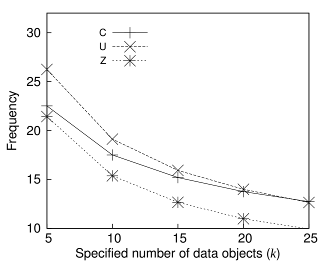


(c) Overlapping Rectangle Attack

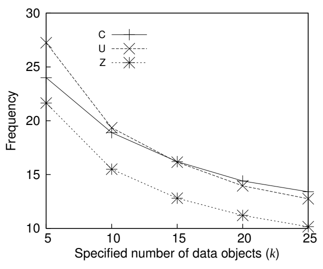


(d) Combined Attack

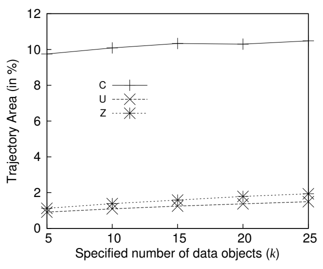
Figure 22: The effect of hiding the required number of NNs on the level of location privacy



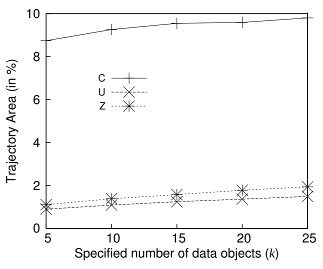
(a) Overlapping Rectangle Attack



(b) Combined Attack

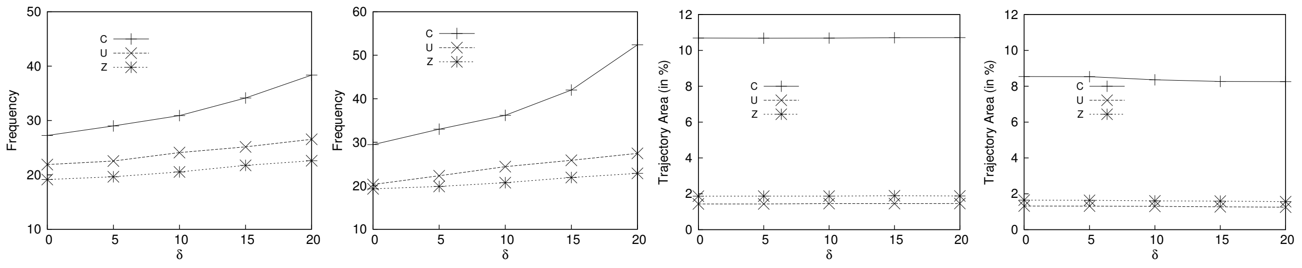


(c) Overlapping Rectangle Attack



(d) Combined Attack

Figure 23: The effect of hiding the specified number of NNs on the level of location privacy



(a) Overlapping Rectangle Attack (b) Combined Attack (c) Overlapping Rectangle Attack (d) Combined Attack

Figure 24: The effect of δ on the level of location privacy

to stricter constraints in the generation of obfuscation rectangle in the combined attack than that of the overlapping rectangle attack.

We also see that a user can achieve a high level of location privacy in terms of frequency by reducing the value of cl_r slightly. For example, in case of the overlapping rectangle attack, the average rate of decrease of frequency are 19% and 10% for reducing the cl_r from 0.9 to 0.8 and from 0.6 to 0.5, respectively, for a fixed $cl = 1$. In case of the combined attack, the average rate of decrease of frequency are 23% and 11% for reducing the cl_r from 0.9 to 0.8 and from 0.6 to 0.5, respectively, for a fixed $cl = 1$. Since the trajectory area almost remains constant for different cl_r, and we can conclude that a user can achieve a high level of location privacy by sacrificing the accuracy of query answers slightly. On the other hand, from Figures 21, we can see that the level of location privacy in terms of both frequency and trajectory area achieves maximum when the specified confidence level is set to 1.

Note that the query processing overhead for a PMkNN query can be approximated by multiplying the frequency for that query with the query processing overhead of single obfuscation rectangle (Section 8.1).

8.2.3. The effect of k_r and k

In these experiments, we observe the effect of the required and the specified number of nearest data objects on the level of location privacy. We vary the value of the required and the specified number of nearest data objects from 1 to 20 and 5 to 25, respectively.

Figures 22(a)-(b) show that the frequency increases with the increase of the required number of nearest data objects k_r for a fixed specified number of nearest data objects $k = 25$. Similar to the case of confidence level, we find that the larger the difference between required and specified number of nearest data objects, the higher the level of location privacy in terms of frequency. On the other hand, Figures 22(c)-(d) show that the trajectory area almost remains constant for different k_r.

Figures 23 show that the frequency decreases and the trajectory area increases with the increase of k for a fixed $k_r = 1$, which is expected as seen in case of confidence level.

Similar to confidence level, we also observe from Figures 22 and 23 that the frequency is higher and the trajectory area is smaller in case of the combined attack than those for the case of the overlapping rectangle attack.

In Figures 23, we also see that the rate of increase of the level of location privacy in terms of both frequency and trajectory area decreases with the increase of k. For example, the highest gain in the level of location privacy for both frequency and trajectory area is achieved when the value of k is increased from 5 to 10. Thus, we conclude that the value of k can be set to 10 to achieve a good level of location privacy for a fixed $k_r = 1$.

8.2.4. The effect of δ

We vary δ from 0 to 20 and find the effect of δ on the level of location privacy in terms of frequency and trajectory area. Figures 24(a)-(b) show that the frequency increases with the increase of δ for both the overlapping rectangle attack and the combined attack. On the other hand, Figures 24(c)-(d) show that the trajectory area almost remains constant for different δ.

9. Conclusions

We have identified the overlapping rectangle attack in an MkNN query and developed a technique that overcomes this attack. Our technique provides a user with three options: if a user does not want to sacrifice the accuracy of answers then the user can protect her privacy by specifying a higher number of NNs than required, which usually comes at a higher cost; alternatively, the user can specify a higher confidence level than required, which incurs a slightly lower accuracy in the query answers; or a user can combine both techniques. We have validated our privacy protection technique with experiments and have found that the larger the difference is between the specified confidence level (or the specified number of NNs) and the required confidence level (or the required number of NNs), the higher is the level of location privacy for M*k*NN queries. We have also proposed an efficient algorithm, CLAPPINQ, which is at least two times faster than Casper and requires at least three times less I/Os.

10. Acknowledgement

This research was supported under Australian Research Council's Discovery Projects funding scheme (project number DP110100757).

References

[1] Microsoft, Location & privacy: Where are we headed?, `http://www.microsoft.com/privacy/dpd` (2011 (accessed September 2, 2011)).

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

[3] S. Nutanong, R. Zhang, E. Tanin, L. Kulik, Analysis and evaluation of V*-*k*nn: an efficient algorithm for moving *k*nn queries, VLDB J. 19 (3) (2010) 307–332.

[4] M. E. Ali, E. Tanin, R. Zhang, K. Ramamohanarao, Probabilistic voronoi diagrams for probabilistic moving nearest neighbor queries, Data Knowl. Eng. 75 (2012) 1–33.

[5] G. Ghinita, M. L. Damiani, C. Silvestri, E. Bertino, Preventing velocity-based linkage attacks in location-aware applications, in: GIS, 2009, pp. 246–255.

[6] M. Xue, P. Kalnis, H. K. Pung, Location diversity: Enhanced privacy protection in location based services, in: International Symposium on Location and Context Awareness, 2009, pp. 70–87.

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

[8] M. Gruteser, D. Grunwald, Anonymous usage of location-based services through spatial and temporal cloaking, in: MobiSys, 2003, pp. 31–42.

[9] R. Cheng, Y. Zhang, E. Bertino, S. Prabhakar, Preserving user location privacy in mobile data management infrastructures, in: Workshop on Privacy Enhancing Technologies, 2006, pp. 393–412.

[10] M. L. Damiani, E. Bertino, C. Silvestri, Protecting location privacy against spatial inferences: the probe approach, in: SIGSPATIAL ACM GIS International Workshop on Security and Privacy in GIS and LBS, 2009, pp. 32–41.

[11] C.-Y. Chow, M. F. Mokbel, Enabling private continuous queries for revealed user locations, in: SSTD, 2007, pp. 258–275.

[12] J. Xu, X. Tang, H. Hu, J. Du, Privacy-conscious location-based queries in mobile environments, IEEE TPDS 99 (1).

[13] T. Xu, Y. Cai, Location anonymity in continuous location-based services, in: ACM GIS, 2007, pp. 1–8.

[14] M. F. Mokbel, C.-Y. Chow, W. G. Aref, The new casper: query processing for location services without compromising privacy, in: VLDB, 2006, pp. 763–774.

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

[16] 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.

[17] M. L. Yiu, C. S. Jensen, J. Møller, H. Lu, Design and analysis of a ranking approach to private location-based services, ACM TODS 36 (2) (2011) 10.

[18] Loopt, `http://www.loopt.com`.

[19] A. Market, `https://market.android.com`.

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

[21] P. Kalnis, G. Ghinita, K. Mouratidis, D. Papadias, Preventing location-based identity inference in anonymous spatial queries, IEEE TKDE 19 (12) (2007) 1719–1733.

[22] T. Hashem, L. Kulik, R. Zhang, Privacy preserving group nearest neighbor queries, in: EDBT, 2010, pp. 489–500.

[23] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, S. Jajodia, Privacy-aware proximity based services, in: MDM, 2009, pp. 31–40.

[24] M. Duckham, L. Kulik, A formal model of obfuscation and negotiation for location privacy, in: Pervasive, 2005, pp. 152–170.

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

[26] A. Khoshgozaran, C. Shahabi, Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy, in: SSTD, 2007, pp. 239–257.

[27] C. Bettini, X. S. Wang, S. Jajodia, Protecting privacy against location-based personal identification, in: VLDB Workshop on Secure Data Management, 2005, pp. 185–199.

[28] A. Gkoulalas-Divanis, V. S. Verykios, M. F. Mokbel, Identifying unsafe routes for network-based trajectory privacy., in: VLDB Workshop on Secure Data Management, 2009, pp. 942–953.

[29] T. Xu, Y. Cai, Exploring historical location data for anonymity preservation in location-based services, in: IEEE INFOCOM, 2008, pp. 547–555.

[30] G. Ghinita, Private queries and trajectory anonymization: a dual perspective on location privacy, Transactions on Data Privacy 2 (1) (2009) 3–19.

[31] N. Li, T. Li, S. Venkatasubramanian, t-closeness: Privacy beyond k-anonymity and l-diversity, in: ICDE, 2007, pp. 106–115.

[32] C. Dwork, J. Lei, Differential privacy and robust statistics, in: STOC, 2009, pp. 371–380.

[33] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, Differential privacy under continual observation, in: STOC, 2010, pp. 715–724.

[34] C. Dwork, The promise of differential privacy: A tutorial on algorithmic techniques, in: FOCS, 2011, pp. 1–2.

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

[36] G. R. Hjaltason, H. Samet, Ranking in spatial databases, in: International Symposium on Advances in Spatial Databases, 1995, pp. 83–95.

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

[38] R. Zhang, H. V. Jagadish, B. T. Dai, K. Ramamohanarao, Optimized algorithms for predictive range and knn queries on moving objects, Inf. Syst. 35 (8) (2010) 911–932.

[39] R. Zhang, D. Lin, K. Ramamohanarao, E. Bertino, Continuous

intersection joins over moving objects, in: ICDE, 2008, pp. 863–872.

[40] R. Zhang, J. Qi, D. Lin, W. Wang, R. C.-W. Wong, Optimized algorithms for predictive range and knn queries on moving objects, VLDB Journal To appear.

[41] S. Saltenis, C. S.Jensen, S. T. Leutenegger, M. A. Lopez, Indexing the positions of continuously moving objects, in: SIGMOD, 2000.

[42] C.-Y. Chow, M. F. Mokbel, W. G. Aref, Casper*: Query processing for location services without compromising privacy, TODS 34 (4) (2009) 24:1–24:48.

[43] C.-Y. Chow, M. F. Mokbel, J. Naps, S. Nath, Approximate evaluation of range nearest neighbor queries with quality guarantee, in: SSTD, 2009, pp. 283–301.

[44] H. Hu, D. L. Lee, Range nearest-neighbor query, IEEE TKDE 18 (1) (2006) 78–91.

[45] L. Kulik, E. Tanin, Incremental rank updates for moving query points., in: GIScience, 2006, pp. 251–268.

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

[47] M. Duckham, L. Kulik, Simulation of obfuscation and negotiation for location privacy, in: Conference on Spatial Information Theory, 2005, pp. 31–48.