Design and Analysis of a Ranking Approach to Private Location-Based Services

MAN LUNG YIU, Hong Kong Polytechnic University
CHRISTIAN S. JENSEN, Aarhus University
JESPER MØLLER, Aalborg University
HUA LU, Aalborg University

Users of mobile services wish to retrieve nearby points of interest without disclosing their locations to the services. The paper addresses the challenge of optimizing the query performance while satisfying given location privacy and query accuracy requirements. The paper's proposal, SpaceTwist, aims to offer location privacy for k nearest neighbor (kNN) queries at low communication cost without requiring a trusted anonymizer. The solution can be used with a conventional DBMS as well as with a server optimized for location-based services. In particular, we believe that this is the first solution that expresses the server-side functionality in a single SQL statement. In its basic form, SpaceTwist utilizes well-known incremental NN query processing on the server. When augmented with a server-side granular search technique, SpaceTwist is capable of exploiting relaxed query accuracy guarantees for obtaining better performance. The paper extends SpaceTwist with so-called ring ranking that improves the communication cost, delayed termination that improves the privacy afforded the user, and the ability to function in spatial networks in addition to Euclidean space. The paper reports on analytical and empirical studies that offer insight into the properties of SpaceTwist and suggest that the paper's proposal is indeed capable of offering privacy with very good performance in realistic settings.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms: Algorithms

Additional Key Words and Phrases: Location Privacy, Mobile Service

1. INTRODUCTION

The mobile Internet offers location-based services (LBSs) that retrieve the locations and other information pertaining to so-called points of interest (data points), e.g., stores, restaurants, and tourist attractions, that are near a user. Some such services rely on the k nearest neighbor (kNN) query [Hjaltason and Samet 1999; Roussopoulos et al. 1995] that retrieves the k data points closest to a user's location q. A server-side database stores a set of points of interest. To retrieve the nearest data point, the user's mobile device sends its location q to the server. The server then computes the (nearest neighbor) result and returns it to the mobile client. As a complication to this scenario, users may wish to avoid disclosing their exact locations to the server.

Author's addresses: M. L. Yiu, Department of Computing, Hong Kong Polytechnic University; email: csmlyiu@comp.polyu.edu.hk; C. S. Jensen, Department of Computer Science, Aarhus University; email: csj@cs.au.dk; J. Møller, Department of Mathematical Sciences, Aalborg University; email: jm@math.aau.dk; H. Lu, Department of Computer Science, Aalborg University; email: luhua@cs.aau.dk.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

A naive private retrieval approach is for the client to download the whole dataset from the server, regardless of the user's location q. Then the client is able to process the query locally. However, the original dataset at the server is subject to updates, so the downloaded file cannot simply be reused for answering future queries at the client. As another drawback, this approach incurs unacceptably high data transmission times, especially for users of mobile devices. Consider a simple experiment that involves a spatial dataset with 500,000 points stored at the server. Each data point consists of an identifier (a 4-byte `integer`), and X, Y coordinates (two 8-byte `double`), so the dataset occupies 9.53 Mbytes. We test with a mobile client device (an Asus P535) with Wi-Fi and GPRS connectivity. We also test with another mobile client device (an HTC Diamond) that supports 3G. The time needed by the client's off-the-shelf Windows Mobile browser for downloading the dataset is 16 s using the fast Wi-Fi connection[1], 37 s using 3G, and 2082 s using GPRS. The high data transfer times of the naive approach make it unattractive.

Existing location privacy approaches can be categorized as spatial anonymization, obfuscation, or private retrieval methods.

Spatial anonymization [Gruteser and Grunwald 2003; Gedik and Liu 2005; Mokbel et al. 2006; Kalnis et al. 2007] employ the architecture shown in Figure 1a. A trusted third-party server, called an anonymizer, is placed in-between the client and the server. The anonymizer enlarges an exact user location q into a (superset) cloaked region Q' so that it contains q and also the locations of $K-1$ other users. This way, the server cannot distinguish q from other user locations in Q'. Upon receiving Q', the server processes a range NN query [Hu and Lee 2006] in order to return a candidate set that contains the nearest data point for any location in Q'. The anonymizer then refines the candidate set and reports the actual result to the client. The advantage of this approach is that it incurs low communication cost between the client and the anonymizer. However, this anonymizer suffers from three drawbacks: (i) it is a performance bottleneck, (ii) it represents a central point of attack (e.g., is prone to collusion attacks from hostile users), and (iii) it heavily depends on the distribution and density of other mobile users.

Obfuscation [Ardagna et al. 2007; Cheng et al. 2006; Xu et al. 2010; Kido et al. 2005; Duckham and Kulik 2005b] adopts the traditional client-server architecture. It avoids the disadvantages associated with the anonymizer. The client is responsible for enlarging the user's location q into an obfuscated set Q'. Specifically, Q' can be represented as a simple region or a discrete set. The server returns the candidate set of Q' to the client, which then computes the actual result from the candidate set. However, this approach also has drawbacks. In case Q' is a simple region, it is hard to control the candidate set size (and the client communication cost), especially when Q' falls into a dense area. On the other hand, if Q' is a discrete set, then it cannot survive location guesses by the adversary; we discuss this in Section 3.2.

Private retrieval methods [Indyk and Woodruff 2006; Khoshgozaran and Shahabi 2007; Ghinita et al. 2008] also operate on the client-server architecture. The client encodes the original query q into an 'incomprehensible' query $q^{\diamondsuit}$. Then the server computes the encoded result of $q^{\diamondsuit}$ blindly. Upon receiving the encoded result, the client derives the actual result. The advantage of this approach is that it offers the strongest privacy guarantee when compared to other approaches. However, it requires specialized algorithms at the client and the server that are hard to implement and do not uti-

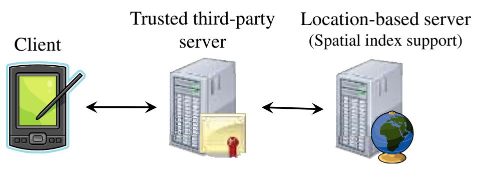
[1]Wi-Fi is of the IEEE 802.11g standard and has a maximum bit rate of 54 Mbit/s. The theoretical data transfer time for the dataset is: 9.53/(54/8)=1.41 s. However, this is unrealistically low as it ignores real-world conditions, e.g., the overhead of communication protocols and the congestion of mobile signals in the multiple-user scenario.

lize existing spatial indexes. They also need a trusted party to perform pre-processing on the dataset beforehand.

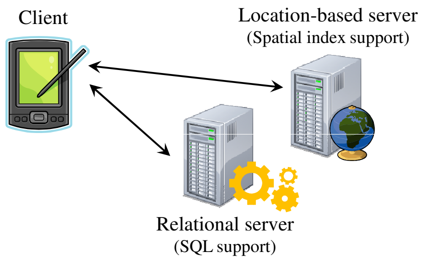
All three kinds of approaches have their own strengths and weaknesses. They are applicable to different scenarios, and none of them can replace the other approaches. Many real-world LBSs (e.g., services in Google's Android Market, Loopt, and Yahoo FireEagle) require users to log in and reveal their identity (User ID). Thus, we are interested in *location privacy* techniques that prevents the server from knowing the user's location (rather than the user's identity) [Jensen et al. 2009]. We adopt the following design requirements for a highly deployable and usable location privacy solution:

— Adopts the conventional client-server architecture (for deployability).
— Leverages existing spatial database technology such as spatial indexes and query processing techniques (for deployability).
— Allows the client to control the communication cost, result accuracy, and privacy (for usability).

Spatial anonymization and private retrieval methods are inapplicable because they do not meet with the first and second requirements, respectively. While obfuscation techniques satisfy these two requirements, most of them do not permit the client to control the communication cost. Our proposed solution belongs to the obfuscation category, and yet it allows the client to control the communication cost conveniently.



(a) Trusted party architecture



(b) Our flexible architecture

Fig. 1. Comparisons of Architectures

Our proposal, called *SpaceTwist*, is a client-server solution (see Figure 1b) that does not require a trusted anonymizer, unlike the spatial anonymization techniques. The client specifies a fake user location called an *anchor*. Then, the server returns data points to the user incrementally in ascending order of their distances from the anchor [Hjaltason and Samet 1999]. The client algorithm processes these data points iteratively until an accurate query result can be reported. In addition, we develop a method that enables the user to control the communication cost conveniently. Furthermore, we also propose several extension techniques that can be integrated into SpaceTwist. They can be used to reduce the communication cost, improve the privacy, and extend the applicability to road-network scenarios.

SpaceTwist is applicable not only to servers optimized for state-of-the-art LBSs, but also to conventional DBMS servers. It leverages existing technology on indexing, query processing and optimization, and other systems aspects. In summary, the paper contributes a client-server location privacy solution that is highly deployable and usable. It provides the following substantial contributions over previous work [Yiu et al. 2008]:

— SQL-based server-side implementation (Sections 3.1, 6.1, and 6.2).

— Analyses and control of the communication cost of SpaceTwist (Section 5).
— A server-side ring ranking technique that reduces the communication cost of exact queries (Section 6.2).
— A delayed termination technique that increases the user's privacy (Section 6.3).
— Applications to spatial networks (Section 6.4).
— Additional empirical studies (Section 7).

Section 2 reviews related work. Section 3 presents the foundations on the incremental nearest neighbor operator and our privacy model. Section 4 develops the query processing technique and analyzes the privacy it affords users. Section 5 presents analyses on the expected communication cost of our solution. Section 6 presents several extensions of SpaceTwist. Section 7 covers the results of extensive studies of the proposed techniques. Finally, Section 8 concludes and identifies research directions.

2. RELATED WORK

Query authentication and privacy are two orthogonal concerns when users issue queries to the server. *Query authentication* [Pang et al. 2005; Li et al. 2006; Yang et al. 2009] focuses on verifying the correctness of the query result returned from an untrusted server. In fact, query authentication on location-based data has also been studied [Yang et al. 2009; Papadopoulos et al. 2009]. On the other hand, *privacy protection* aims at preventing the server from knowing the user's original query information (e.g., identity, location).

We categorize existing privacy protection approaches into: spatial anonymization, obfuscation, and private retrieval methods. Spatial anonymization requires an anonymizer in the architecture, whereas the other two approaches adopt the (conventional) client-server architecture.

2.1. Spatial Anonymization

Spatial anonymization techniques [Gruteser and Grunwald 2003; Mokbel et al. 2006; Kalnis et al. 2007] aim at protecting the *identity privacy* [Bettini et al. 2007] of users. The adversary is assumed to know the exact locations of all users in the system. The K-anonymity model [Sweeney 2002] aims at preventing the adversary from identifying a user (from the user's query) with probability above $1/K$.

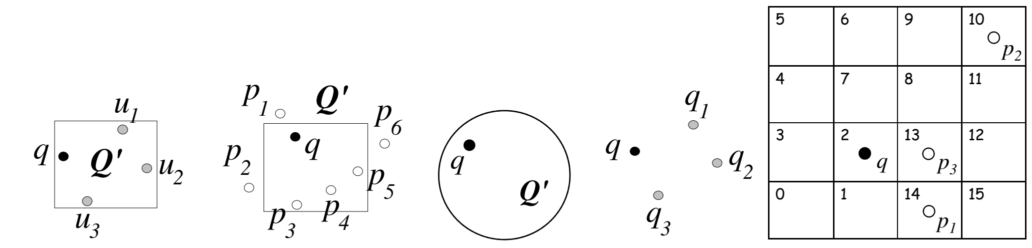
To achieve this, a trusted third-party location anonymizer [Gruteser and Grunwald 2003; Mokbel et al. 2006; Kalnis et al. 2007] is employed to maintain the current locations for all users. When a user issues a query, the anonymizer computes a K-anonymous region Q' that contains the user's location q and at least $K-1$ other user locations. This prevents the adversary from identifying the user from Q with probability above $1/K$. Figure 2a illustrates a 4-anonymous region Q', where u_1, u_2, and u_3 are $(4-1)$ user locations.

Instead of using q, the anonymizer sends the cloaked query Q' to the server for processing. Observe that the cloaked query cannot be answered by point-based kNN algorithms [Roussopoulos et al. 1995; Hjaltason and Samet 1999]. Specialized server-side algorithms [Mokbel et al. 2006; Hu and Lee 2006; Kalnis et al. 2007] are needed for identifying a candidate set that includes the kNN for any location in a cloaked region. In the example of Figure 2b, the candidate set of Q' contains these points: $p_1, p_2, p_3, p_4, p_5, p_6$; they can become NN for any location in Q'. Specifically, Mokbel et al. [2006] propose efficient heuristics for computing a superset of the candidate set, Hu and Lee [2006] develop the range kNN algorithm for computing the minimal possible candidate set for a rectangular cloaked query, and Kalnis et al. [2007] study the computation of the candidate set for a circular cloaked query. After receiving the candidate set from the server, the anonymizer derives the actual result for q and sends the result to the user.

In addition to supporting the K-anonymity model, the anonymizer can be extended to support alternative privacy requirements, such as minimum cloaked area [Mokbel et al. 2006], maximum cloaked area [Bamba et al. 2008], location diversity [Bamba et al. 2008], and sensitive area hiding [Gruteser and Liu 2004].

The advantage of spatial anonymization is that the communication cost between the client and the anonymizer is optimal. Unfortunately, the anonymizer and the K-anonymity model exhibit several disadvantages. First, the anonymizer becomes a performance bottleneck because it needs to serve all its subscribed users, as well as maintaining accurate records of their locations. Second, the anonymizer is vulnerable to malicious attacks. An example is the *collusion attack*, where the adversary subscribes to the anonymizer multiple times with fake user locations. Then, the adversary can exclude those fake locations from the cloaked region Q' generated by the anonymizer, this way identifying a user with higher probability. Third, it heavily depends on the distribution and density of other mobile users. When the user is located in a sparse region or few users are subscribed to the anonymizer, it needs to construct a very large cloaked region Q' such that it contains K user locations. This incurs high processing time at the server and the anonymizer.

Alternatively, a K-anonymous region can be derived through peer-to-peer communication [Chow et al. 2006; Ghinita et al. 2007b; 2007a]. Users close together form a group and set their cloaked region as a rectangle containing them. The drawback is that group formation and maintenance incur communication latency.



(a) K-anonymity (b) range kNN (c) obfuscation (d) dummies (e) Hilbert retrieval

Fig. 2. Examples of Location Privacy Solutions

2.2. Obfuscation

In the *location privacy* model, the adversary (e.g., the server) knows the user's identity (the user's ID), but not the user's location. Obfuscation techniques [Cheng et al. 2006; Xu et al. 2010; Ardagna et al. 2007; Kido et al. 2005; Duckham and Kulik 2005b; Lu et al. 2008] aim to protect the user's exact location from being revealed. Based on given privacy requirements, the client enlarges the user's exact location q into an obfuscated set Q' that contains q. This approach does not require any anonymizer.

The obfuscated set Q' can be represented as a *connected obfuscated region* (see Figure 2c) or a *discrete obfuscated set* (see Figures 2d).

Connected Obfuscated Query.
A connected obfuscated query Q' is a simple region (e.g., a rectangle or circle) that contains the user's location q. An example is shown in Figure 2c. It can be processed by server-side techniques employed in K-anonymization solutions. For instance, the processing of a rectangular query and a circular query can be handled by the proposal

of Hu and Lee [2006] and Kalnis et al. [2007], respectively. The only difference is that no anonymizer is used now. Upon receiving the candidate results from the server, the client is responsible for refining them into the actual result. The disadvantage of using a connected obfuscated region Q' is that it is hard to control the communication cost, especially when Q' overlaps with a dense region (of data points).

Various techniques have been proposed for the client to construct Q' from q.

The study of Ardagna et al. [2007] takes location positioning inaccuracy into account, models the user location as a circular region, and develops several geometric operators for deriving obfuscated regions.

Xu et al. [2010] study the obfuscation for a continuously-moving query user. Each obfuscated region is a circle whose area is required to be above a user-specified value A_{min}. Let $Q(t')$ be the previous obfuscated region, $Q(t)$ be the current obfuscated region, and D^* be the maximum possible travel distance of the user (from time t' to time t). By exploiting this information, the adversary can launch the *trace analysis attack* to deduce the probabilistic density distribution of the user within region $Q(t)$. The entropy value can be derived from the above probabilistic density distribution. It is zero when the user is at a location with probability 1; it is maximized when the user is at any location in $Q(t)$ with the same probability. To defend against the *trace analysis attack*, Xu et al. [2010] formulate a set of linear equations for constructing a current region $Q(t)$ that maximizes the entropy. They also introduce two objective functions for optimizing the mobile client's result accuracy or communication cost. The equation set may not always have a feasible solution; in such cases, the query is *blocked*, and the candidate set of the previous query $Q(t')$ is used to compute an approximate result instead of sending a new query to the server. This method imposes considerable computational overhead during obfuscation at the client, which needs to solve the above equations using linear programming techniques together with discretization. In our problem setting, where we consider the snapshot kNN query (i.e., only issued by a user once), the above trace analysis attack is not relevant, and the obfuscation method of Xu et al. [2010] is degraded to a simple obfuscation method that generates any obfuscated region around the actual user's location q. Furthermore, the proposal of [Xu et al. 2010] relies on linear programming techniques in Euclidean space and so cannot be adapted to road networks, unlike our method.

Cheng et al. [2006] process range queries on a private dataset that contains the (obfuscated) locations of all users. The client enlarges the user's exact location q into a circular region Q' based on the user's requirement on the region area and the coverage of sensitive facilities (e.g., a hospital). The server manages the circular regions of all users. Upon receiving a range query, the server computes a candidate result set and derives the probability/confidence of each candidate being an actual result. The quality of the result set is summarized by a quality score that combines the confidence value of each candidate. This method is inapplicable to our work, which uses a public dataset (e.g., locations of restaurants) and considers kNN queries. Cheng et al. [2006] do not study techniques for reducing the communication cost between the server and the query client; and probabilistic results are returned—not the exact results as required in our work. Although the method of Cheng et al. [2006] can be generalized to the case where all other users' locations are precise points, each candidate only qualifies as result for part of the cloaked range query so a candidate is still associated with a probability.

Discrete Obfuscated Query.
A discrete obfuscated query Q' contains the user's location q and a number of dummy locations [Kido et al. 2005; Duckham and Kulik 2005a; 2005b; Lu et al. 2008]. An example [Kido et al. 2005] is shown in Figure 2d, in which q_1, q_2, and q_3 are dummy locations,

and the obfuscation is $Q' = \{q, q_1, q_2, q_3\}$. Enhanced techniques have also been studied for generating dummy locations, e.g., based on the distances among the locations [Lu et al. 2008]. In another proposal, the possible locations are restricted to be vertices in a graph (e.g., a road network), and the obfuscation is a set of vertices [Duckham and Kulik 2005a; 2005b].

The obfuscation Q' can be answered by processing each point in the query in turn, returning the union of the results [Kido et al. 2005; Lu et al. 2008]. Alternatively, an interactive negotiation protocol has been proposed that enables on-line trade-offs between privacy and result accuracy [Duckham and Kulik 2005a; 2005b].

A discrete obfuscated set Q' allows easy control of the communication cost. However, it is not location-collision-resistant, so it may not survive location guesses by the adversary. We elaborate on this issue in Section 3.2.

2.3. Private Retrieval Methods

Private retrieval methods [Indyk and Woodruff 2006; Khoshgozaran and Shahabi 2007; Ghinita et al. 2008] use the client-server architecture. The client encodes its original query q into an 'incomprehensible' query $q^{\diamondsuit}$. Next, the server computes the encoded result of $q^{\diamondsuit}$ blindly, and then the client derives the actual result from the encoded result. This approach offers the strongest privacy guarantee when compared to spatial anonymization and obfuscation approaches. However, it requires specialized algorithms at the client and the server that are hard to implement and do not utilize existing spatial indexes.

A theoretical study of a client-server protocol for deriving the nearest neighbor of q has been reported [Indyk and Woodruff 2006]. Its communication cost is asymptotic to $\sqrt{N}$, where N is the number of data points. It does not necessarily report the exact result. No experimental evaluation of the communication cost and result accuracy of the protocol with real data are available.

Another study defines a specific Hilbert ordering based on a key $\mathcal{H}$, whose value is known only by the client and a trusted entity [Khoshgozaran and Shahabi 2007]. Not having the key value, the server cannot decode a Hilbert value into a location correctly. In preparation for querying, the trusted entity transforms each data point p_i into a Hilbert value $\mathcal{H}(p_i)$ that is uploaded to the server. In the example of Figure 2e, the Hilbert value of each cell is shown in the top-left corner of the cell. At query time, the client q submits its Hilbert value $\mathcal{H}(q) = 2$ to the server. The server then reports the Hilbert value $\mathcal{H}(p_2) = 10$ that is closest to $\mathcal{H}(q)$. This Hilbert value is eventually decoded by the client into point p_2. Observe that a Hilbert curve does not completely preserve spatial proximity, so the reported result can be far from q. To improve the accuracy, the use of two keys $\mathcal{H}$ and $\mathcal{H}'$ with orthogonal Hilbert curves has been considered [Khoshgozaran and Shahabi 2007]. Still, this enhanced method does not necessarily report the exact result.

Ghinita et al. [2008] study private information retrieval methods for answering the NN. Their proposal includes an approximate method and an exact method. Both are built upon a computationally secure protocol that retrieves a number from a two-dimensional array. In their approximate method, the data points are partitioned into $\sqrt{N}$ leaf nodes of a $\mathbb{K}$-d tree [Bentley 1975], where N is the number of data points. In their exact method, the space is divided into $G \times G$ grid partitions (where G is a parameter) and the Voronoi diagram of the dataset is pre-computed. Each partition stores the data points whose Voronoi cells intersect the grid cell. For both methods, during query time, the user requests all data points of the (server-side) partition that covers the query point, then derives locally the result from those points. However, the above methods suffer from two limitations. First, they are applicable only to the NN query, not to kNN queries. For instance, the exact solution utilizes a Voronoi diagram, which cannot be used to answer kNN queries. Second, the exact method leads to very high execution times at the server side (20 seconds for a single-CPU server [Ghinita et al. 2008]), meaning that the server cannot handle a high volume of queries within a short time period.

2.4. Cloaking and Query Processing in Road Networks

We proceed to review solutions that provide location privacy to users on a road network. Duckham and Kulik [2005a; 2005b] provide the first study of location privacy when the query and data points are constrained to a spatial network such as a road network. In their solution, the server and the client agree on a negotiation protocol for retrieving the results for the user. During execution, the user may be required to make multiple interactive decisions on whether or not to release a more accurate obfuscation set for the user's location. A reasonably accurate result can only be obtained if the user accepts to reveal a sufficiently accurate obfuscation set. The protocol cannot achieve high accuracy and high privacy simultaneously.

Other works adopt the K-anonymity model and focus on protecting the identity privacy of users [Ku et al. 2007; Li et al. 2008; Mouratidis and Yiu 2010; Wang and Liu 2009]. An anonymizer is employed to construct a cloaked query Q' such that it contains the user's location q and $K-1$ other user locations. To process the cloaked query Q', the server executes three steps: (i) compute the border intersections between Q' and the road network, (ii) find the kNNs for each such intersection, and (iii) fetch all data points that fall into Q'. The server then returns the union of the results from steps (ii) and (iii) as the candidate result set to the anonymizer. This approach exhibits two drawbacks. First, the processing cost depends on the number of border intersections between Q' and the road network, which becomes high for a large Q'. Second, duplicates are found in step (ii), wasting computational effort. To avoid the above limitations, our network-based SpaceTwist (in Section 6.4) directly executes the network-based incremental nearest neighbor algorithm [Papadias et al. 2003] at the server side. This guarantees that no duplicates are retrieved. Also, our solution is easy to implement, as it does not perform low-level operations (e.g., computing border intersections).

3. FOUNDATIONS

This section presents the foundations for the subsequent sections. We first cover the implementation of the incremental nearest neighbor operator [Hjaltason and Samet 1999] in Section 3.1 and then propose our privacy model in Section 3.2. Table I summarizes the notation to be used throughout the paper.

Table I. Summary of Notation

Symbol	Meaning
q	the actual user location
q'	the anchor location
P	the set of points of interest/data points
p_i	a data point of P
$dist(q, p_i)$	Euclidean distance between the points q and p_i
$dist_G(q, p_i)$	network distance between the points q and p_i
N	the cardinality of P
k	the number of requested nearest neighbors
m	the number of received points
Ψ	the inferred privacy region
$\Gamma(q, \Psi)$	the privacy value
ϵ	the guaranteed error for relaxed kNN

3.1. Implementation of the Incremental Nearest Neighbor Operator

Let q' be a given anchor location. Given a set P of data points, the *incremental nearest neighbor* (INN) operator retrieves each point $p \in P$ in ascending order of $dist(q', p)$, i.e., the Euclidean distance between q' and p.

As the INN operator is an important component of our solutions, we proceed to discuss the implementations of the INN operator with different types of servers.

Implementation on a Server for Location-Based Services (LBSs).
This kind of server is designed with the objective of processing location-based queries. For efficient query processing, the server usually employs a spatial index (e.g., an R-tree) for indexing a point set P. Hjaltason and Samet [1999] develop an I/O-efficient implementation of the INN operator on an R-tree that indexes P.

Implementation Using a Conventional DBMS.
A DBMS is a general-purpose system that supports a wide range of applications. In order to facilitate easy deployment on a DBMS and also significantly reduce the development cost, SQL has been used to compute various queries: approximate string join [Gravano et al. 2001], skyline retrieval [Bartolini et al. 2006], kNN queries and kNN joins [Yao et al. 2010]. Past works use plain SQL queries as opposed to user-defined functions (UDF). The latter generally do not integrate well with query optimizers and incur expensive computation overhead. Particularly, in the proposal of Bartolini et al. [2006], the client issues an SQL `ORDER BY` query to retrieve tuples progressively from the DBMS server until the client is guaranteed to compute the exact result (of a skyline query). In the same spirit, we study how to implement the INN operator on top of a relational DBMS.

First, we propose to store the point set P as a *relational table* with schema (id, x, y), where id is an identifer, x and y are location coordinates. Each row corresponds to a point p in P. Then, we denote the coordinates of the anchor location q' by $\_qx'$ and $\_qy'$ (where an underscore indicates a given value). The distance $dist(q', p)$ can then be expressed as $\sqrt{(x - \_qx') \cdot (x - \_qx') + (y - \_qy') \cdot (y - \_qy')}$. Thus, we can implement the INN operator by the following SQL query.

```
SELECT      id, x, y
FROM        P
ORDER BY    (x-_qx')·(x-_qx') + (y-_qy')·(y-_qy')     ASC
```

The `ORDER BY` clause uses the squared distance because it preserves the ordering of distances and is faster to compute than $dist(q', p)$. The query is a ranking query, and it can be executed efficiently by the DBMS server by means of various optimization strategies [Ilyas et al. 2004; Ilyas et al. 2006; Xin et al. 2006; Xin et al. 2007; Zou and Chen 2008].

More efficient implementations of kNN search in relational databases are provided in the literature [Jagadish et al. 2005; Yao et al. 2010]. These employ a relational table with schema (id, x, y, key) where key is an additional attribute that captures partial location information. A primary index (e.g., a clustering B^+-tree) is built on the attribute key in order to support efficient access to the table. Jagadish et al. [2005] set the key of a point to be a composite value that consists of its nearest cluster ID and its distance to the cluster center, and Yao et al. [2010] set the key of a point to be a space filling-curve value (e.g., the z-value of the point). At query time, the server first retrieves k points whose key are closest to that of q. Then, the server executes a range search with a certain key range on the table to retrieve all candidate points that can become results. These candidates are then filtered in order to obtain the final results.

We believe that these methods can be adapted for incremental nearest neighbor search by iteratively expanding k and excluding the key range of reported result points.

Transferring Results from the Server to the Client.
We propose an efficient *Non-blocking* implementation that enables the client to terminate the transfer of data points before having received all the points returned by the SQL query (or by the LBSs). Once the client initiates the server with a `Start` message, the server sends incremental NNs to the client in a streaming manner, without the need for explicit `GetNext` requests from the client. When the client decides to terminate the process, it sends a `Stop` message to the server. Let m be the number of points received by the client at this time. Before the server encounters the `Stop` message, the client will receive $\beta^* = m'' - m$ *unnecessary data points* from the server, where m'' is the total number of points sent by the server. Details are available in Appendix A. Our empirical testing on real mobile devices reveal that the measured value of β^* fluctuates within a certain technology-dependent range $[\beta^-, \beta^+]$. As the actual value of β^* cannot be predicted, the adversary can only infer the possible range of m to be: $[m'' - \beta^+, m'' - \beta^-]$. This information will be considered in our privacy analysis in Section 4.2.

Furthermore, in Section 5.6, we explore how to reduce the execution time of DBMS by estimating a tight bound on the number of tuples to be retrieved.

3.2. Privacy Model

Adversary Model and Privacy Quantization.
Before formulating the concept of privacy, we capture the capabilities of the adversary via an *adversary model*. This is in line with the location privacy model adopted in obfuscation techniques (see Section 2).

Many real-world LBSs (e.g., services available in Google's Android Market and from Loopt and Yahoo's FireEagle) require users to register and log in as prerequisites for using services. Thus, the users reveal their identity (User ID), and identity privacy is not an issue in our problem setting, rendering the K-anonymity model [Sweeney 2002; Mokbel et al. 2006] inapplicable. We assume that services only access the content of the messages sent to and from the users, and we assume that those location-based services cannot collude with the communication providers (e.g., telecom companies and Internet service providers) of the users. Thus, the location-based server is unable to infer the user's location via other means such as wireless signal strengths, hands-off signals, and IP addresses. Several IP address geo-location tools[2] exist; however, IP addresses only allow coarse (e.g., city-level) positioning, and they may even point to the locations of Internet service providers. Such information is not useful for the adversary, for typical location-based applications on a city map. Alternatively, users can hide their IP addresses from the LBS server by using a proxy server or distributed anonymous networking software[3].

Based on the above observation, our adversary is assumed to have the user's exact identity, but no knowledge of the user's current location q. We then capture the *worst case scenario* in our adversary model, i.e., the adversary has complete knowledge of: (i) the messages sent between the client and the server, (ii) the algorithms executed at the client and the server, and (iii) the parameters used by those algorithms (except the user's current location). The goal of the adversary is then to infer the user's *location* from the query issued by the user.

[2]http://ipinfodb.com; http://www.ip2location.com
[3]http://www.torproject.org

We proceed to give a generic definition of a user's *privacy region* Ψ in Definition 3.1. We highlight that Ψ can be given explicitly as a connected, rectangular region [Gruteser and Grunwald 2003; Mokbel et al. 2006; Kalnis et al. 2007], or can be derived implicitly based on the adversary's knowledge (as for SpaceTwist in Section 4.2).

Definition 3.1. The privacy region Ψ of a user is defined as a set such that: Ψ contains the user's exact location q, and the adversary cannot distinguish q from other locations in Ψ.

Next, we introduce an essential concept called *location collision privacy*. In this model, the adversary is given κ (an integer) resource units and allowed to pick a set Ψ' of κ distinct points from the privacy region Ψ. The region Ψ is said to be unsafe if the adversary's picked set Ψ' contains the user's exact location q. We then formulate this definition as follows.

Definition 3.2. A region Ψ is said to be *location-collision-resistant* if for any positive integer κ, there exists a set Ψ' of κ points such that $q \notin \Psi'$ and $\Psi' \subset \Psi$.

Observe that any connected privacy region (say, Ψ_{conn}) is collision-resistant because, for any κ, we can always pick κ distinct points from Ψ_{conn} such that they are different from q. In contrast, any discrete privacy region (say, Ψ_{dis}) is not collision-resistant because it fails to satisfy the condition of Definition 3.2 when $\kappa = |\Psi_{\mathrm{dis}}|$. E.g., the dummy query in Figure 2d cannot survive from 4 location guesses by the adversary. Thus, we eliminate discrete privacy region (e.g., dummies) from further consideration in this paper.

We attempt to define the privacy value of the user based on the proximity between the actual user location q and other locations in Ψ. Intuitively, the privacy value of Ψ is high if most of the adversary's picked locations (from Ψ) are located far away from q. Thus, we quantify the *privacy value* as the average distance from a location (uniformly distributed) in Ψ to the user's actual location q:

Definition 3.3. Let q be a user location and Ψ be a privacy region. The *privacy value* $\Upsilon(q, \Psi)$ of q and Ψ is given as follows:

$$\Upsilon(q, \Psi) = \frac{\int_{z \in \Psi} dist(z, q)\, dz}{\int_{z \in \Psi}\, dz} \tag{1}$$

Extension for Advanced Constraints and Preferences.
The above definition of the privacy value is built on the basic assumption that each location in the inferred region Ψ is equally probable of being the actual user location. Our privacy model can be adapted to take into account complex features: (i) spatial domain constraints (e.g., excluding low density regions such as forests and lakes from the space) and (ii) user preferences (e.g., a user requires low privacy at work and high privacy when visiting a clinic). In such scenarios, we employ a weighting function $\omega : \mathbb{R}^2 \to \mathbb{R}_+$. The value of $\omega(z)$ can be set to a high value when the location z warrants additional privacy, e.g., is a clinic, the user's home, or another sensitive location. With the function $\omega(\cdot)$, we define the *conditional privacy value* $\Upsilon_{cond}(q, \Psi)$ of q as follows.

$$\Upsilon_{cond}(q, \Psi) = \frac{\int_{z \in \Psi} dist(z, q) \cdot \omega(z)\, dz}{\int_{z \in \Psi} \omega(z)\, dz} \tag{2}$$

4. SPACETWIST: INCREMENTAL PROCESSING

We propose an algorithm that computes exact kNN query results in an incremental fashion while affording the user location privacy. We assume only a simple client-server architecture, and we assume that the server indexes the dataset P by an R-tree [Guttman 1984] and supports incremental nearest neighbor retrieval [Hjaltason and Samet 1999]. We consider the snapshot kNN query [Mokbel et al. 2006; Kalnis et al. 2007], leaving continuously-moving queries [Chow and Mokbel 2007; Xu et al. 2010] for future work.

In this paper, we assume that the server evaluates queries correctly and it does not report altered result to the client. This issue is orthogonal to our study, and it can be achieved by existing query authentication method [Yang et al. 2009] that verifies the correctness of the query result reported from an untrusted server. These methods consider the owner O_P of the dataset P, who builds an *authenticated index* on the dataset, before distributing it to the server. Upon receiving a query, the server not only computes the query result, but also derives a *verification object* from the authenticated index as a proof of the result correctness. The client then verifies the correctness of the query result based on the verification object. In our problem setting, the dataset P is public (e.g., restaurant locations) and its authoritative data owner O_P could be the government's business registration department, which is usually an entity different from the server. For the sake of generality, we also consider the case where the data owner is the same as the server. In this scenario, the server could request a certification authority (CA) to build the authenticated index on P. The client trusts the CA and uses its public key for the verification process.

We now offer an overview of our technique in Figure 3. Instead of sending the actual user location q to the server, the client sends an *anchor* (a "fake" location) q' and then iteratively requests data points from the server in ascending distance order [Hjaltason and Samet 1999] from the anchor. The *supply space* centered at the anchor is the part of space already explored. The *demand space* denotes the space to be covered before the client is guaranteed to be able to produce an accurate result. The client knows both the demand space and the supply space, whereas the server knows only the supply space. In the beginning (see Figure 3a), the demand space equals the domain space, and the supply space is empty. As points are retrieved from the server, the supply space expands. When a retrieved point p is the closest point to the client seen so far, the results are updated, and the demand space shrinks. When the supply space eventually covers the demand space (see Figure 3b), it is termed *final*, and the client is guaranteed to be able to produce an accurate result. The *communication cost* of our solution is measured as the total number of points received by the client — a platform-independent measure. Detailed discussion on the transmission of results can be found in Section 3.1 (and Appendix A).

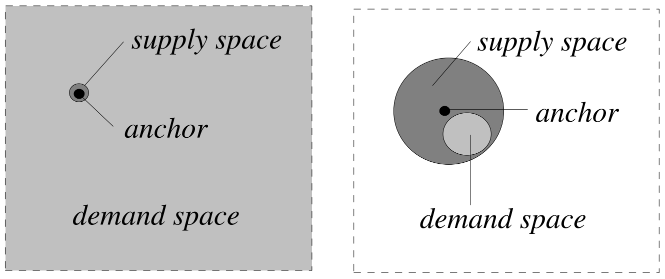
Following the above overview, Section 4.1 describes the client-side algorithm. Section 4.2 analyzes the location privacy achieved, and Section 4.3 presents a visualization of the privacy region by means of primitive geometric shapes.

4.1. The SpaceTwist Client-Side Algorithm

We proceed to present the client-side algorithm for accurate kNN retrieval. We use the notation $dist(q, p)$ to denote the Euclidean distance between points q and p.

The client (i.e., the user) executes Algorithm 1 to obtain its k nearest objects from the server (i.e., the query processor). The *anchor location* q' is first sent to the server. The user's actual location q is known only by the client. Intuitively, if q and q' are close then few objects are retrieved (i.e., low cost), but less location privacy is achieved. We will present a procedure for selecting an appropriate q' in Section 5.5.

A max-heap W_k, initialized with k virtual objects, maintains the k nearest objects (of q) seen so far. Let γ be the maximum distance in W_k. The *demand space* is then the circle with radius γ and center q (see Line 3). Let τ be the largest distance between q' and any object examined so far. The *supply space* is then the circle with radius τ



(a) the beginning (b) the end

Fig. 3. Demand Space and Supply Space

Algorithm 1 Space Twist Client (for kNN query)

```
algorithm SpaceTwistClient(Value $k$, Point $q$, Point $q'$)
1:  $W_k \leftarrow$ new max-heap of pairs $\langle p, dist(q,p)\rangle$;
2:  insert $k$ pairs of $\langle NULL, \infty\rangle$ into $W_k$;
3:  $\gamma \leftarrow$ the top distance of $W_k$;          ▷ $k^{th}$ best distance from $q$
4:  $\tau \leftarrow 0$;                                    ▷ furthest distance seen from $q'$
5:  send an INN query with $q'$ to the server;
6:  while $\gamma + dist(q,q') > \tau$ do
7:      $p \leftarrow$ get the next point from the server;
8:      $\tau \leftarrow dist(q',p)$;                       ▷ update supply space
9:      if $dist(q,p) < \gamma$ then                       ▷ check demand space
10:         update $W_k$ (and $\gamma$) by using $p$;
11: terminate the INN query at the server;
12: return $W_k$;
```

and center q' (see Line 4). Next, the server is requested to return incremental nearest neighbors (INNs) of q'.

In Line 7, the client retrieves the next INN (of q') from the server. The distance τ is updated to be the furthest distance between q' and the retrieved point p. We then check whether $dist(q,p)$ is less than γ (i.e., whether q is closer to p than some object in W_k). If so, then W_k and γ are updated. According to Lemma 4.1, the loop continues as long as $\gamma + dist(q,q') > \tau$. Finally, the client returns the result set W_k after terminating the INN query at the server.

LEMMA 4.1. **Termination condition.**
If $\gamma + dist(q,q') \le \tau$ then the k^{th} nearest object (say, $p_\star$) of q has been retrieved.

PROOF. Since the upper pound of the distance between $p_\star$ and q is γ, the upper bound on its distance to q' is $\gamma + dist(q,q')$, according to the triangular inequality. Based on the property of incremental nearest neighbor retrieval [Hjaltason and Samet 1999], all objects within distance τ from q' have been seen. Thus, we conclude that $p_\star$ has already been retrieved. □

Example.
Figure 4 exemplifies the algorithm for the case $k = 1$. When we discover point p_1 (see Figure 4a), we set the best result to p_1 and define the demand space (light gray area) around q as well as the supply space (dark gray area) around q'. Next, in Figure 4b, point p_2 is discovered and the supply space expands. Since q is closer to p_2 than the previous result (i.e., p_1), the best result is updated to be p_2 and the demand space

shrinks. Then point p_3 is retrieved (see Figure 4c), and the supply space grows. As the supply space encloses the demand space, the algorithm terminates and returns p_2 as the nearest neighbor of q.

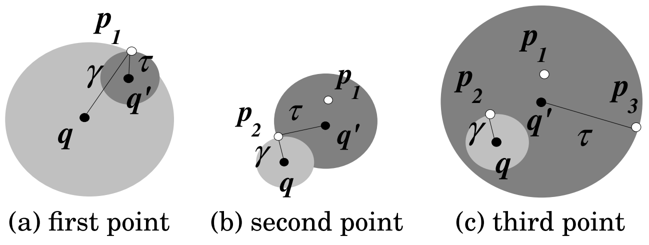


Fig. 4. Query Processing Example

We observe that, based on the given parameters k, q, and q', the algorithm computes the *exact result* and terminates without requesting unnecessary points from the server, due to Lemma 4.1.

4.2. Privacy Analysis

This section studies how an adversary is able to *infer* the possible locations of the user. We assume that the adversary knows: (i) the anchor q' and the value k, (ii) the points reported by the server, and (iii) the termination condition of Algorithm 1. As we assume a simple client-server architecture, the concept of K-anonymity [Sweeney 2002; Mokbel et al. 2006] is inapplicable.

Privacy Properties of SpaceTwist.
We denote a *possible* user location by q_c in order to distinguish it from the *actual* user location q. For now, we consider q_c as a random variable of q while fixing the parameters k and q' at some given values.

Note that the anchor q' deterministically defines the order in which data points are retrieved from the server via the INN query. Thus, the SpaceTwist client-side algorithm is deterministic, meaning that the same input parameter q always yields exactly the same sequence of points received from the server. The number of received points must be an integer between 1 and N.

Since SpaceTwist is deterministic, we can conceptually define a function $\aleph(q_c) : \mathbb{R}^2 \rightarrow [1, N]$ that returns the number of points received for the query point q_c. We use this for defining the privacy region $\Psi_{q',k}(m)$ as the set of all possible query locations such that exactly m points are retrieved from the server:

$$\Psi_{q',k}(m) = \{q_c \in \mathbb{R}^2 \mid \aleph(q_c) = m\}$$

Observe that $\Psi_{q',k}(m)$ constitutes a partitioning of the space domain for different values of m. In other words, $\bigcup_{m\in[1,N]} \Psi_{q',k}(m)$ is equal to the space domain, and $\Psi_{q',k}(m)$ and $\Psi_{q',k}(m')$ are disjoint for $m \neq m'$.

In the following, we discuss how to formulate the region $\Psi_{q',k}(m)$ by utilizing inequalities.

Formulation of Privacy Region by Inequalities.
Let m be the number of points received by the client and let the points received in the order of their retrieval be $p_1, p_2, \ldots, p_m$. Since the algorithm did not terminate at the

final point received, we have:

$$dist(q_c, q') + \overset{k}{\min_{1 \le i \le (m-1)}} dist(q_c, p_i) > dist(q', p_{(m-1)}) \ , \tag{3}$$

where the middle term represents the k^{th} smallest distance of the first $(m-1)$ points from q_c.

The adversary knows the last point leading the algorithm to terminate. Thus, the adversary deduces:

$$dist(q_c, q') + \overset{k}{\min_{1 \le i \le m}} dist(q_c, p_i) \le dist(q', p_m) \tag{4}$$

Clearly, a possible user location q_c must satisfy both inequalities above.

While region Ψ can be inferred by both the user and the adversary, only the user can derive the privacy value $\Upsilon(q, \Psi)$, as q is required in Equation 1.

Since Ψ does not have closed-form expression in the general case, its derivation is non-trivial. Monte Carlo methods can be used for approximating Ψ, by randomly generating candidate locations for q_c and checking them against inequalities 3 and 4.

Recall from Section 3.1 (and Appendix A) that in the Non-blocking implementation of the incremental NN search, m represents the number of points received by the client and m'' represents the total number of points sent by the server. Further, we defined $\beta^* = m'' - m$ be their difference. In general, the adversary, as an observer of the server, knows only m'' and an estimate of β^* (in the range $[\beta^-, \beta^+]$), as discussed in Section 3.1 (and Appendix A. Instead of knowing the exact m value, the adversary can only deduce the possible range of m to be: $[m'' - \beta^+, m'' - \beta^-]$. In order to capture this scenario, we replace $m-1$ by $m'' - \beta^+ - 1$ and m by $m'' - \beta^-$ in Equations 3 and 4, respectively.

Exact Privacy Region Derivation.
Fortunately, we have found a closed-form expression for Ψ for the case of $k = 1$. For each retrieved point p_i, we can derive $Vor(p_i)$, its Voronoi cell [Okabe et al. 2000] with respect to all retrieved points. Observe that p_i is the NN of any location q_c inside $Vor(p_i)$. Furthermore, the possible location of q_c is constrained by the termination condition of the algorithm. Figure 5a depicts the final supply space as the circle with radius $dist(q', p_m)$ and center at the anchor q'. Termination occurs when the supply space covers the demand space:

$$dist(q_c, q') + dist(q_c, p_i) \le dist(q', p_m)$$

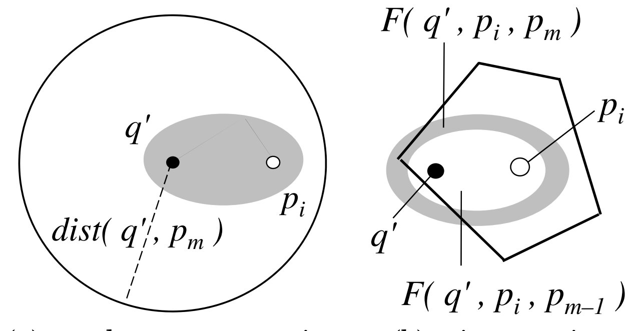
Definition 4.2. **Elliptical region.**
Given the anchor point q' and two retrieved points p_i and p_j (where $i < j$ in the retrieval order), we define $F(q', p_i, p_j)$ as an elliptical region such that (i) its foci are q' and p_i, and (ii) any point on the border of the elliptical region has its sum of distances to the foci being equal to $dist(q', p_j)$.

The set of locations satisfying this inequality can be expressed as an elliptical region $F(q', p_i, p_m)$ (shown in gray in Figure 5a) with foci q' and p_i, where any point on the border has its sum of distances to the foci being equal to $dist(q', p_m)$.

Similarly, we obtain $F(q', p_i, p_{m-1})$, the elliptical region with p_m replaced by p_{m-1}. Since the algorithm did not terminate at point p_{m-1}, we exclude $F(q', p_i, p_{m-1})$ from the possible region, as shown in Figure 5b.

Combining the above with the Voronoi cell $Vor(p_i)$, the inferred privacy region is given by:

$$\Psi = \bigcup_{i=1}^{m} Vor(p_i) \cap (F(q', p_i, p_m) \setminus F(q', p_i, p_{m-1}))$$

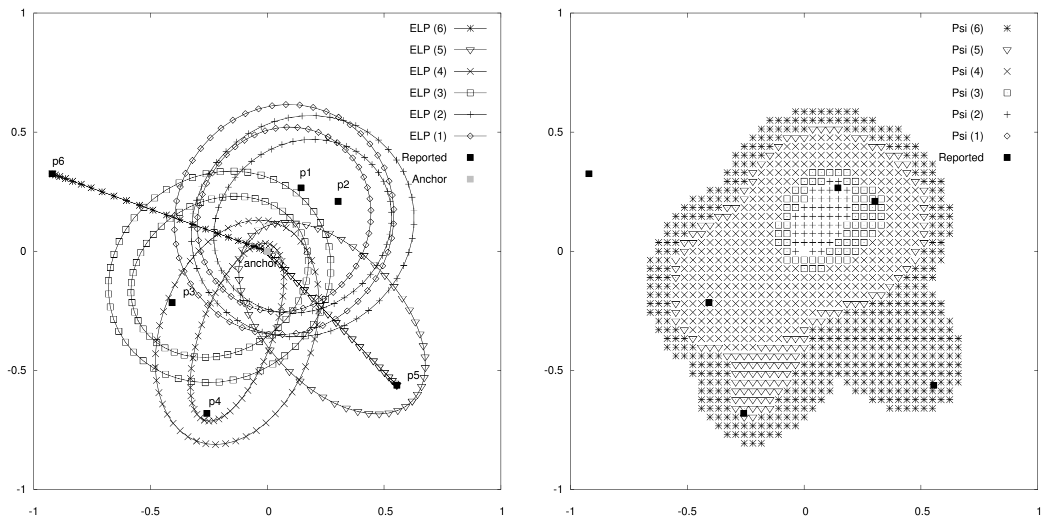


(a) supply space constraint (b) privacy region

Fig. 5. Inferring a Privacy Region

4.3. Visualization of Privacy Region

We apply the above derivation to a dataset in order to visualize the inferred privacy region Ψ. Figure 6a shows the anchor location q' and $m = 6$ retrieved points $p_1, p_2, \ldots, p_6$, whose associated ellipses are shown using the symbols $\Diamond$, $+$, $\Box$, $\times$, $\triangledown$, and $*$, respectively.



(a) elliptical regions, $m = 6$ (b) privacy regions, $m \in [1, 6]$

Fig. 6. Visualization of Inferred Privacy Regions

Following the notation from Section 4.2, each point p_i is used to derive its outer elliptical region $F(q', p_i, p_m)$ and its inner elliptical region $F(q', p_i, p_{m-1})$. For instance, the point p_3 is used to derive its outer ($\Box$) elliptical region $F(q', p_3, p_6)$ and inner ($\Box$) elliptical region $F(q', p_3, p_5)$. By taking the intersection of the region $F(q', p_3, p_6) \setminus F(q', p_3, p_5)$ with the Voronoi cell of p_3 (not shown here), we obtain the set of possible user location q_c such that it takes p_3 as its nearest neighbor and causes the algorithm to terminate after receiving p_6.

It is worth noticing that for point p_5, the inner ($\triangledown$) elliptical region $F(q', p_5, p_5)$ degenerates to a line. Also, for point p_6, the outer ($*$) elliptical region $F(q', p_6, p_6)$ degenerates to a line, whereas its inner elliptical region $F(q', p_6, p_5)$ is undefined.

Figure 6b depicts, for each value $m \in [1, 6]$, the corresponding inferred privacy region when the algorithm terminates after receiving the m-th point. For instance, the region marked by $*$ corresponds to the inferred privacy region at $m = 6$, and it resembles the shapes of elliptical regions shown in Figure 6a. All possible query locations in that privacy region have the same behavior from the perspective of the server—they trigger the algorithm to terminate after receiving point p_6. In Figure 6b, the inferred privacy region at $m = 5$ (4, 3, 2, 1) is indicated by regions marked by different symbols, as listed in the figure's legend.

5. COMMUNICATION COST ANALYSIS

We proceed to analyze the communication cost incurred by SpaceTwist. Our objective is to derive the expected number of points m reported by the server to the client. This enables the selection of an appropriate location for the anchor q' that satisfies a given communication cost budget.

We first employ a stochastic process for concisely representing the distribution of data points in Section 5.1, next apply a radial simulation technique to model the behavior of SpaceTwist in Section 5.2, and then derive the expected communication cost in Section 5.3. Our stochastic analysis is restricted to the case with parameter values $k = 1$ and a 2D point dataset with a uniform distribution.

For any arbitrarily distributed dataset, we exploit a spatial histogram for estimating the communication cost in Section 5.4, and then present an efficient client-side anchor selection procedure based on communication cost budget in Section 5.5. We also discuss how to utilize our cost model for limiting the server cost of the SQL implementation of SpaceTwist in Section 5.6.

5.1. Definitions and Properties

For the sake of our analysis, we first introduce notation for the areas of a region and a circular region. Recall that $\mathbb{R}$ represents the set of real numbers and $\mathbb{R}^2$ denotes the two-dimensional space with real-valued coordinates. We use $|\mathcal{BR}|$ to represent the area of a bounded region $\mathcal{BR} \subset \mathbb{R}^2$. Let $\odot(w, r)$ be the circular region with center w and radius r. Formally, we have:

$$\odot(w, r) = \{w' \in \mathbb{R}^2 \mid dist(w', w) \leq r\}$$

Several data distributions are needed for the analysis. We denote the uniform distribution on a bounded set $A \subset \mathbb{R}$ by $\mathrm{Uni}(A)$; the exponential distribution on the positive half-line $(0, \infty)$ by $\mathrm{Exp}(\mu)$, where $1/\mu > 0$ is the mean value; the gamma distribution on $(0, \infty)$ by $\Gamma(\alpha, \gamma)$, where $\alpha > 0$ is the shape parameter and $\gamma > 0$ is the inverse scale parameter (i.e., the mean value is α/γ, and $\Gamma(1, \mu) = \mathrm{Exp}(\mu)$); and the beta distribution on $(0, 1)$ by $B(\alpha_1, \alpha_2)$, where $\alpha_1 > 0$ and $\alpha_2 > 0$ are the shape parameters.

We say $X \sim \mathcal{D}$ when the random variable X has the probability distribution $\mathcal{D}$. For instance, the expression $U_i \sim \mathrm{Uni}((0, 1))$ means that the random variable U_i has the uniform probability distribution on the range $(0, 1)$.

In probability theory, a *planar point process* is a random countable subset of $\mathbb{R}^2$. Throughout the analysis, we adopt the stationary Poisson process, which is a natural and basic point process that can be utilized for constructing more advanced point process models [Møller and Waagepetersen 2004]. In addition, this model eliminates boundary effects, which renders the subsequent mathematical analysis easier to understand.

In particular, we consider an infinite set of data points $\Phi = \{p_1, p_2, p_3, \ldots\}$ in $\mathbb{R}^2$ that conforms the *stationary Poisson model* with an intensity parameter $\rho > 0$. This model has the following properties [Kingman 1993]:

— For any bounded region $\mathcal{BR} \subset \mathbb{R}^2$, the points in $\Phi \cap \mathcal{BR}$ are independently and uniformly distributed in $\mathcal{BR}$.
— For any bounded region $\mathcal{BR} \subset \mathbb{R}^2$, the expected number of points in $\Phi \cap \mathcal{BR}$ is $\rho \cdot |\mathcal{BR}|$. This number follows a Poisson distribution.
— The model is *stationary*, meaning that the distribution of Φ is invariant under arbitrary translation in $\mathbb{R}^2$.
— The model is *isotropic*, meaning that the distribution of Φ is invariant under arbitrary rotation at q' in $\mathbb{R}^2$.

Let $l = dist(q, q')$ be the distance between the anchor q' and the user location q. Due to the stationarity and isotropy of Φ, we set q and q' as follows, without loss of generality.

$$q' = (0,0), \quad q = (l, 0) \tag{5}$$

This simplifies the mathematical exposition to follow.

5.2. Radial Simulation Technique

We proceed to discuss a simulation technique that generates the elements of set Φ such that it follows the stationary Poisson distribution. Specifically, Quine and Watson [1984] propose a radial simulation technique for this purpose. The process of radial simulation and its property are captured by the following theorem.

THEOREM 5.1. *Consider independent random variables $\theta_1, U_1,\ \theta_2, U_2, \ldots$, where $\theta_i \sim \mathrm{Uni}([0, 2\pi))$ and $U_i \sim \mathrm{Uni}((0,1))$. We set*

$$S_i = -\frac{1}{\pi\rho} \ln U_i, \quad i = 1, 2, \ldots, \tag{6}$$

where $\ln$ *denotes the natural logarithm, and we let*

$$R_0 = 0, \quad R_i = \sqrt{R_{i-1}^2 + S_i}, \quad i = 1, 2, \ldots.$$

Then $p_i = (R_i \cos\theta_i, R_i \sin\theta_i)$, $i = 1, 2, \ldots$, constitute the radially ordered points of a stationary Poisson process with intensity ρ.

By definition the sequence of radii R_i is increasing (and strictly increasing with probability one), and the squared radii R_i^2 form a homogeneous Poisson point process on the positive half-line $(0, \infty)$, and this point process is independent of the point process of the angular coordinates $\theta_1, \theta_2, \ldots$. Moreover, the S_i are independent and identically distributed, with $S_i \sim \mathrm{Exp}(\pi\rho)$.

The following lemma shows the properties of the first m data points generated by the radial simulation technique, namely that these m points considered without their radial ordering are independent and uniformly distributed in the region $\odot(q', R_{m+1})$. Thus, the (first m) points generated by the radial simulation technique effectively capture the characteristics of the points returned by the SpaceTwist client-side algorithm.

COROLLARY 5.2. **Properties of radial simulation technique.**
Following straightforwardly from Theorem 5.1, we have the following properties for any $m \in \mathbb{N}$:

(*1*) $R_m^2 \sim \Gamma(m, \pi\rho)$ *is independent of* $\theta_m \sim \mathrm{Uni}([0, 2\pi))$.
(*2*) *Conditional on $R_{m+1} = r$, the set $\{p_1, p_2, \ldots, p_m\}$ is a binomial point process, i.e., these m unordered data points are independent and uniformly distributed in the region $\odot(q', r)$.*

5.3. Stochastic Communication Cost Analysis

We are now able to analyze the communication cost of SpaceTwist (see Algorithm 1). For the moment, we consider both q and q' as fixed points.

Let the point p_j be the point accessed by the algorithm in its j^{th} iteration. We define the current supply space as:

$$\mathcal{S}_j = \odot(q', R_j) \ ,$$

where $R_j = dist(p_j, q')$ is the distance between the last received point and q'. Then we define the current demand space as:

$$\mathcal{D}_j = \odot(q, \tilde{R}_j) \ ,$$

where $\tilde{R}_j = \min_{1 \le i \le j} dist(p_i, q)$ is the (smallest) nearest neighbor distance from q seen so far.

By Corollary 5.2, with probability 1, the sequence $R_1, R_2, \ldots$ increases strictly towards infinity, causing the sequence of supply spaces to expand, i.e., $\mathcal{S}_1 \subset \mathcal{S}_2 \subset \ldots$. Further, the sequence $\tilde{R}_1, \tilde{R}_2, \ldots$ decreases, so the sequence of demand spaces shrinks, i.e., $\mathcal{D}_1 \supseteq \mathcal{D}_2 \supseteq \ldots$. The communication cost M is then given by:

$$M = \min\{j \in \mathbb{N} \mid \mathcal{D}_j \subseteq \mathcal{S}_j\}$$

The capital letter M is used to stress that this is a random variable that depends on the points q and q' and the distribution of the data points. Our task is to derive the distribution of M and in particular its expected value $\mathrm{E}(M)$ and variance $\mathrm{V}(M)$. The following is a special case of results derived in Møller and Yiu [2010].

THEOREM 5.3. *The communication cost M has probability distribution*

$$\mathrm{P}r(M = m) = \int_0^\infty \frac{\left(\alpha^2 + 2\alpha\sqrt{s}\right)^{m-2}}{(m-2)!} \mathrm{e}^{-\left(\alpha^2 + 2\alpha\sqrt{s} + s\right)} \, \mathrm{d}s, \quad m = 2, 3, \ldots \tag{7}$$

and mean and variance given by

$$\mathrm{E}(M) = \alpha^2 + \sqrt{\pi}\alpha + 2, \quad \mathrm{V}(M) = (5 - \pi)\alpha^2 + \sqrt{\pi}\alpha,$$

with $\alpha = \sqrt{\pi\rho}\, l$.

PROOF. See Theorem 1 in Møller and Yiu [2010]. □

Let $\mathrm{erf}(\alpha) = (2/\sqrt{\pi}) \int_0^\alpha \exp(-t^2) \, \mathrm{d}t$ be the 'error function'. Then Equation 7 gives

$$\mathrm{P}(M = 2) = \exp(-\alpha^2) + \alpha\sqrt{\pi}(\mathrm{erf}(\alpha) - 1),$$

which strictly decreases from one to zero as α decreases from zero to infinity. We have also evaluated the integral in Equation 7 for $m = 3, 4, \ldots$ using the computational software program Maple, but since the number of terms increases fast as m increases, we omit the results here.

5.4. Communication Cost for Arbitrary Data Distributions

In this section, we utilize a spatial histogram to capture the distribution of the dataset P. Based on the histogram, we develop a client-side technique that estimates the communication cost for a given query location q and an anchor location q'.

The distribution of a point dataset P can be summarized concisely by a *spatial histogram* [Acharya et al. 1999], which partitions the domain space into bins. Each bin represents a spatial region and records the count of points in P that fall into its region. Figure 7 shows a simple equi-width spatial histogram with 5×5 bins. The count of each bin is also shown in the figure. For instance, the top-right-most bin has the rectangle $[0.8, 1.0) \times [0.8, 1.0)$ as its spatial region and has count 5.

Spatial histograms have been applied for the estimation of the result sizes of spatial range queries. The so-called MinSkew histogram [Acharya et al. 1999] incurs smaller estimation errors than other histograms, given the same number of bins. It is thus the state-of-the-art histogram.

Let $\mathcal{H}$ be the spatial histogram of the dataset P. To facilitate the estimation procedure, the client needs to request the histogram $\mathcal{H}$ from the server. This incurs only small communication overhead as the typical number of bins in $\mathcal{H}$ is small.

Figure 8a illustrates how the communication cost of SpaceTwist is estimated for the given anchor q'. Let $\widehat{\gamma}$ be the estimated kNN distance of the original query location q, i.e., the distance from q to its k^{th} nearest neighbor in P. Its value can be estimated from $\mathcal{H}$ by applying the estimation technique of Tao et al. [2004]. When SpaceTwist terminates, the supply space covers the demand space. Thus, we derive $\widehat{\tau} = \widehat{\gamma} + dist(q, q')$ as the estimated supply space radius. Let $\odot(q', \widehat{\tau})$ be the circle centered at the anchor q' and with radius $\widehat{\tau}$. Let $\widehat{m}$ be the estimated number of points in the circle $\odot(q', \widehat{\tau})$, i.e., estimated communication cost. Its value can be estimated from $\mathcal{H}$ by the estimation technique of Acharya et al. [1999]. The above estimation procedure is summarized in Lines 6–8 of Algorithm 2.

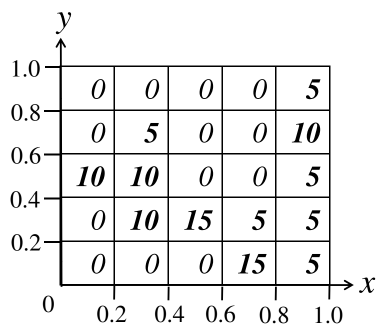
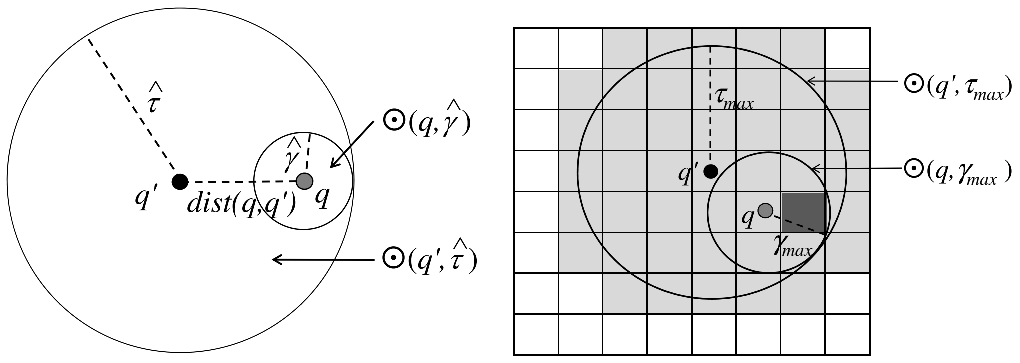


Fig. 7. Spatial Histogram



(a) cardinality estimation (b) limiting cost

Fig. 8. Using a Spatial Histogram for Cost Estimation

Algorithm 2 Cost-Estimation

```
    algorithm Cost-Estimation(Value $k$, Point $q$, Budget $m^*$)
 1: $q' \leftarrow$ a random location;
 2: $dist_{lb}^{anc} \leftarrow 0$;
 3: $dist_{ub}^{anc} \leftarrow$ the maximum distance of the spatial domain;
 4: retrieve the spatial histogram $\mathcal{H}$ of the dataset $P$ from the server;
 5: repeat
 6:     $\widehat{\gamma} \leftarrow$ the estimated $k$NN distance of $q$;        ▷ Apply [Tao et al. 2004] on $\mathcal{H}$
 7:     $\widehat{\tau} \leftarrow \widehat{\gamma} + dist(q, q')$;
 8:     $\widehat{m} \leftarrow$ the estimated cardinality in $\odot(q', \widehat{\tau})$;   ▷ Apply [Acharya et al. 1999] on $\mathcal{H}$
 9:     if $\widehat{m} < m^*$ then
10:         $dist_{lb}^{anc} \leftarrow dist(q, q')$;
11:     else if $\widehat{m} > m^*$ then
12:         $dist_{ub}^{anc} \leftarrow dist(q, q')$;
13:     $d' \leftarrow (dist_{lb}^{anc} + dist_{ub}^{anc})/2$;
14:     $q' \leftarrow q + (q' - q) \cdot \frac{d'}{dist(q,q')}$;
15: until $|\widehat{m} - m^*| \leq \delta$
16: return the point $q'$;
```

5.5. Anchor Generation Based on a Communication Cost Budget

Next, we use the technique from the previous section for generating a suitable anchor q' based on a given communication cost budget m^*. Specifically, we present a client procedure for generating an anchor q' such that its estimated number of retrieved points $\widehat{m}$ is sufficiently close to m^*, i.e., the absolute difference between $\widehat{m}$ and m^* is below a certain threshold δ.

This problem can be solved by applying the bisection method in multiple rounds—see Algorithm 2. Initially, the client picks a random location q', then sets the lower-bound anchor distance $dist_{lb}^{anc}$ to 0, and sets the upper-bound anchor distance $dist_{ub}^{anc}$ to the maximum distance in the spatial domain.

In each round, the client runs Algorithm 2 to obtain the estimated cost $\widehat{m}$. If $\widehat{m}$ is smaller than m^*, the client updates $dist_{lb}^{anc}$ to $dist(q, q')$. If $\widehat{m}$ is greater than m^*, the client updates $dist_{ub}^{anc}$ to $dist(q, q')$. After that, the client moves q' towards/away from q by updating $dist(q, q')$ to $\frac{dist_{lb}^{anc}+dist_{ub}^{anc}}{2}$. The above process is repeated until the estimated cost is sufficiently close to the budget m^*.

As a remark, Algorithm 2 is efficient, and convergence occurs in 10 iterations.

Correctness and Privacy When Using a Spatial Histogram.
The algorithm just presented needs to retrieve the spatial histogram $\mathcal{H}$ of the dataset P from the server. This prompts two new questions:

— How does the client detect whether the server returns a genuine histogram?
— Can the server exploit the histogram and Algorithm 2 to deduce the user's location?

We consider each question in turn in two settings. In the first, the spatial dataset originates from a separate data owner (see the beginning of Section 4). In this setting, the data owner can compute the histogram and attach a digital signature to it before sending it to the server. When the client receives the histogram from the server at query time, it can easily verify the correctness of the histogram by using its digital signature and the data owner's public key.

In the second setting, the histogram is built by the server, and no digital signature is available. We then assume that the adversary is aware of the approximate location of the client q. As this can be difficult for the adversary to obtain, this is a worst-case assumption. The adversary could then manufacture dense histogram buckets near q.

Due to the budget constraint, the client tends to choose a location for q' that is close to q, so this may reveal considerable location information for q. This problem can be detected as follows. When the client retrieves fewer points than the estimated number from the histogram, it regards the server as malicious and can then apply the delayed termination technique (to be discussed in Section 6.3). The client then keeps retrieving points from the server until its budget is spent, in order to obfuscate its actual location.

Considering the second question, we find that the histogram does not help the server deduce the user's actual location q. As described in Section 4.2, after the execution of SpaceTwist, the server already knows the anchor point q', the value of k, and the communication cost m^*. The server can then derive the inferred privacy region $\Psi_{q',k}(m^*)$ (see Equation 3), which represents the set of all possible query locations such that the server returns exactly m^* points. Even if the server knows the histogram and Algorithm 2, the server still cannot eliminate any possible query location from $\Psi_{q',k}(m^*)$, according to Lemma 5.4.

LEMMA 5.4. **Privacy of anchor generation.**
Given the value of k, the communication cost budget m^*, the generated anchor q' of Algorithm 2, and the spatial histogram $\mathcal{H}$, any location q_c in $\Psi_{q',k}(m^*)$ could have been the actual query location q.

PROOF. By the definition of $\Psi_{q',k}(m^*)$ (see Equation 3), any possible location $q_c \in \Psi_{q',k}(m^*)$ causes SpaceTwist to return m^* points. Observe that Algorithm 2 generates an anchor q' from the actual query point q, so that its estimated cost on the histogram $\mathcal{H}$ equals the communication cost budget m^*. Thus, any possible location $q_c \in \Psi_{q',k}(m^*)$ could have been the input query location q of Algorithm 2. $\square$

5.6. Limiting the Cost of the SQL Implementation

Recall that our server-side functionality can be implemented by a SQL query. The computational effort of the server can be significantly reduced by including the "LIMIT m" clause[4] into the SQL query of Section 3.1, where m is (an upper bound on) the number of data points to be retrieved by the client.

To achieve this, we use a spatial histogram to compute a tight upper-bound on m. Our technique consists of three steps. Consider the spatial histogram of Figure 8b as an example and let $k = 1$. First, the client finds the bin such that its maximum distance γ_{max} to q is minimized. That bin is shown in dark gray. Observe that the circle $\odot(q, \gamma_{max})$ is guaranteed to enclose the actual NN of q. Second, the client formulates the supply space as the circle $\odot(q', \tau_{max})$, where $\tau_{max} = \gamma_{max} + dist(q', q)$. Third, the client finds the total count of bins (shown in light gray) whose extents intersect with $\odot(q', \tau_{max})$. This total count is used as an upper bound on the communication cost m. It guarantees that all points within $\odot(q', \tau_{max})$ will be retrieved and thus that the client will compute the correct result.

The above technique is applicable to arbitrary k, by modifying the first step to find the subset of bins such that: (i) the sum of their counts is at least k, and (ii) their maximum distance to q is minimized.

6. EXTENSIONS

We present four extensions to SpaceTwist. Section 6.1 studies a granular search technique that reduces the communication cost by relaxing the result accuracy. Section 6.2 presents an incremental ring ranking technique that aims to reduce the communication cost without compromising the result accuracy. Section 6.3 proposes a delayed

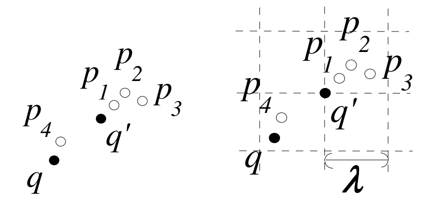
[4]Different RDBMS products provide this functionality using slightly different syntax, e.g., "TOP" for MS-SQL Server, "LIMIT" for MySQL, and "ROWNUM" for Oracle.

termination strategy capable of enhancing the privacy value in a dynamic manner. Section 6.4 applies SpaceTwist to querying data points constrained to a road network.

6.1. Granular Search

We proceed to equip SpaceTwist with a server-side granular search technique that is capable of retrieving data points from the server with a user-specific granularity. This technique enables communication cost reduction and location privacy improvement while providing strict guarantees on the accuracies of the query results. First, we describe granular search for the case $k = 1$. Then we examine its implementation and extension to arbitrary k. Finally, we discuss how to implement granular search using SQL on a conventional DBMS.

Recall that the client-side algorithm requests data points from the server in ascending order of their distance to anchor q'. For the example in Figure 9a, the server returns points in the order: p_1, p_2, p_3, p_4. Although p_4 is the actual NN of q, this point cannot be obtained early by the client.



(a) set of points (b) grid cells

Fig. 9. Granular Search

The communication cost can be reduced by returning only a sample of the reported data points. A threshold ϵ is then introduced for controlling the result accuracy:

Definition 6.1. **ϵ-relaxed kNN query.**
Given a point set P, a distance threshold ϵ, and a location q, a point $p \in P$ is said to be an ϵ-relaxed kNN of q when $dist(q,p) \leq \epsilon + \min^k_{p' \in P} dist(q,p')$, where the last term represents the distance between q and its k^{th} NN in P.

The idea behind granular search is to impose a grid (with cell extent λ) on the domain space, as shown in Figure 9b. When the server iteratively retrieves incremental nearest neighbors of anchor q', it disregards points in a grid cell from which a point has already been reported. To ensure that the query result is an ϵ-relaxed NN of q, it suffices to set the cell extent λ to $\epsilon/\sqrt{2}$, as shown in the lemma below.

LEMMA 6.2. **Result distance guarantee.**
Consider a regular grid with cell extent λ. Let $p_\star$ be the actual NN of q and p' be the retrieved NN of q. It holds that $dist(q,p') \leq dist(q,p_\star) + \sqrt{2} \cdot \lambda$.

PROOF. In case $p_\star$ has been retrieved, the inequality holds trivially (by setting p' to $p_\star$).

Otherwise, $p_\star$ has not been retrieved. Thus, a point p'' in the cell of $p_\star$ must have been retrieved. The maximum possible distance between p and p'' is the diagonal length of the cell, i.e., $\sqrt{2} \cdot \lambda$. From the triangular inequality, we obtain $dist(q,p'') \leq dist(q,p_\star) + dist(p_\star,p'') \leq dist(q,p_\star) + \sqrt{2} \cdot \lambda$. Since p' is the retrieved NN of q, we have $dist(q,p') \leq dist(q,p'')$, completing the proof. □

Continuing with the example in Figure 9b, the server first sends point p_1 to the client. Since p_2 and p_3 fall in the cell of p_1, they are disregarded. Finally, p_4 is reported to the client. In this example, the communication cost drops from 4 to 2 data points.

We provide the following guidelines for the client to choose the values for ϵ and q'. It is intuitive to set the error bound $\epsilon = v_{max} \cdot \Delta t_{max}$, according to the maximum speed v_{max} of the user and the maximum travel time delay Δt_{max} acceptable by the user. For instance, a typical value for Δt_{max} may be 5 minutes and the value of v_{max} depends on the user's mode of transportation (e.g., walking, bicycling, driving).

The anchor q' can be determined by the algorithm presented in Section 5.4. The only difference is that we need to use a spatial histogram that captures the data distribution of the points considered by the granular search.

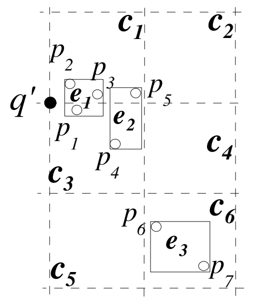
Granular Search Using an LBS Server.
We proceed to consider the implementation of the above method. Algorithm 3 shows our granular incremental NN algorithm, which takes the user-specified error bound ϵ as input. A conceptual grid with cell extent λ ($= \epsilon/\sqrt{2}$) is imposed on the returned points during runtime. The algorithm also takes an R-tree R (indexing the data points) and an anchor q' as arguments. The notation $mindist(q', e)$ ($maxdist(q', e)$) represents the minimum (maximum) possible distance between q' and an R-tree entry e [Roussopoulos et al. 1995; Hjaltason and Samet 1999]. Next, $\mathcal{C}_\lambda(p)$ denotes the cell containing point p.

Algorithm 3 Granular Incremental NN

```
    algorithm GranularINN(R-Tree $R$, Point $q'$, Value $\epsilon$)
 1: $\lambda \leftarrow \epsilon/\sqrt{2}$;
 2: $H \leftarrow$ new min-heap ($mindist$ to $q'$ as key);
 3: $V \leftarrow$ new set;                                  ▷ cells of reported points
 4: for all entries $e \in R.root$ do
 5:     insert $\langle e, mindist(q', e)\rangle$ into $H$;
 6: while $H$ is not empty do
 7:     deheap $\langle e, mindist(q', e)\rangle$ from $H$;
 8:     remove each cell $c$ from $V$ satisfying
         $maxdist(q', c) < mindist(q', e)$;
 9:     if $e$ is not covered by the union of cells in $V$ then
10:         if $e$ is a point $p$ then
11:             report $p$ to the client;
12:             $V \leftarrow V \cup \{\mathcal{C}_\lambda(p)\}$;
13:         else
14:             read the child node $CN'$ pointed to by $e$;
15:             for all entries $e' \in CN'$ do
16:                 insert $\langle e', mindist(q', e')\rangle$ into $H$;
```

The algorithm applies INN search to anchor q', with two modifications: (i) a set V is employed (Line 3) for tracking the grid cells of the reported points (Line 12), and (ii) only qualifying entries that are not covered by the union of cells in V are processed further (Line 9).

Figure 10b illustrates the use of granular NN search on the example in Figure 10a. An R-tree is assumed, the root of which contains the three entries e_1, e_2, and e_3, each of which points to a leaf node. Each cell is marked by a bold label c_i. The algorithm first examines the root of the R-tree, inserting entries e_1, e_2, and e_3 into heap H. Next, e_1 is deheaped and its child entries p_1, p_2, and p_3 are inserted into H. Then, p_1 is found and reported, and its corresponding cell c_3 is added to V. Next, p_2 is found and reported, and its cell c_1 is inserted into V. When point p_3 is deheaped, it is discarded because it falls into a cell (i.e., c_1) in V. Similarly, entry e_2 is discarded, as it is covered by the union of the cells c_1 and c_3 in V. After that, e_3 is deheaped, and its child entries p_6, and p_7 are inserted into H. Cell c_1 (and c_3) is removed from V, as it cannot intersect any point or entry encountered in the future. The algorithm continues until H becomes empty or it is terminated by the client.



(a) locations of points

Operation	**Heap** H	**Cell list** V
Expand root	e_1, e_2, e_3	$\varnothing$
Expand e_1	p_1, p_2, p_3, e_2, e_3	$\varnothing$
Report p_1	p_2, p_3, e_2, e_3	c_3
Report p_2	p_3, e_2, e_3	c_3, c_1
Discard p_3	e_2, e_3	c_3, c_1
Discard e_2	e_3	c_3, c_1
Expand e_3	p_6, p_7	$\varnothing$
Report p_6	p_7	c_6
Discard p_7	$\varnothing$	c_6

(b) execution steps

Fig. 10. Granular INN Example

We observe that Lemma 6.2 can be extended to kNN search as well. The basic idea is to keep not just one, but k points in each cell. To accomplish this, Algorithm 3 is modified as follows. First, each cell $c \in V$ is associated with a counter $cnt(c)$. Second, in Line 12, we check whether the cell $\mathcal{C}_\lambda(p)$ already exists in V. If so, we increment its counter; otherwise, we insert the cell (with counter value 1) into V. Third, in Line 9, we only consider the cells with a counter value of k.

Granular Search Using a Conventional DBMS.
We consider the implementation of granular search on a conventional DBMS.

Let the coordinates of the anchor q' be denoted by $\_qx'$ and $\_qy'$. Let the value of λ be $\_lambda$. We propose the following SQL query for performing granular search. For each data point $p \in P$, the coordinates of its grid cell are represented by gx and gy, respectively. Then we group the points based on the values of gx and gy. Next, we order the groups in ascending order of their distances from the anchor. Finally, the `Top-k` is used to return (at most) k points from each group.

```
SELECT      TOP-k(id)
FROM        P
GROUP BY    ROUND(x,_lambda) AS gx, ROUND(y,_lambda) AS gy
ORDER BY    (gx-_qx')·(gx-_qx')+(gy-_qy')·(gy-_qy') ASC
```

6.2. Communication Cost Reduction: Incremental Ring Ranking

If the exact result of the kNN query is needed, the technique of Section 6.1 cannot be applied. Here, we propose a new server-side ranking technique that reduces the communication cost for exact queries. This technique is inspired by an observation that can be made from Figure 6b. The inferred privacy region is an irregular, ring-shaped region around the anchor q'. This suggests that efforts to retrieve points near q' will be in vain because the adversary can learn that the user is not located at the "center" of the ring.

This suggests that it may be possible to develop a technique for ordering the points, such that (i) it produces inferred privacy regions with extents similar to those produced by the original SpaceTwist algorithm, and (ii) it retrieves only the "necessary" points close to the inferred privacy regions.

A New Server-Side Retrieval Order.
We name this technique *incremental ring ranking*. The client is required to specify two input parameters: an anchor location q' and a ring radius R. An optimal communication cost can be achieved by setting $R = dist(q, q')$; but other values of R may be specified by the user.

The server now retrieves data points p in ascending order of their distance $|dist(q', p) - R|$. This can be achieved by a slightly modified incremental nearest neighbor algorithm [Hjaltason and Samet 1999]. Specifically, the key of each heap entry e (in the min-heap H) is set to the value $|mindist(q', e) - R|$ instead of the original distance $mindist(q', e)$, where q' is the anchor.

Modifications of the SpaceTwist Client.
Due to the new retrieval order, the client-side algorithm (Algorithm 1) also needs modification (in Lines 4, 5, and 8) such that its correctness is guaranteed.

First, the ring radius R is included as a parameter. It defines a ring that is a (hollow) circle centered at anchor q' and with radius R. In Line 4, the variable τ is used to keep the furthest distance of retrieved points from the ring. In Line 5, we invoke the incremental ring ranking method on the server side using parameters q' and R. In Line 8, we set τ to the absolute value $|dist(q', p) - R|$.

Lemma 6.3 states the termination condition of the modified client-side algorithm, which guarantees the correctness of the result. The condition in Line 6 is replaced by the negated termination condition: $(R - \tau > dist(q', q) - \gamma)$ or $(R + \tau < dist(q', q) + \gamma)$.

LEMMA 6.3. **Termination condition of ring-based retrieval.**
If $R - \tau \leq dist(q', q) - \gamma$ and $R + \tau \geq dist(q', q) + \gamma$ then the actual k^{th} nearest object (say, $p_\star$) of q has been retrieved.

PROOF. The distance $dist(q', q) - \gamma$ denotes the minimum possible distance of $p_\star$ from q', whereas the distance $dist(q', q) + \gamma$ denotes the maximum possible distance of $p_\star$ from q'. Based on the property of incremental ring-based retrieval, all objects within the distance range $[R - \tau, R + \tau]$ from q' have been seen. Since $R - \tau \leq dist(q', q) - \gamma$ and $R + \tau \geq dist(q', q) + \gamma$, $p_\star$ has already been retrieved. □

As a corollary of Lemma 6.3, the termination condition is simplified to $\gamma \leq \tau$, for the special case $dist(q', q) = R$.

Example.
Figure 11 illustrates the modified SpaceTwist algorithm for $k = 1$. The user initially submits the parameter values q' and R to the server, requesting data points incrementally in ring-based order.

In Figure 11a, the point p_1 is received from the server. Value τ is set to the distance of p_1 from the ring (whose radius is R). The search space is a (dark gray) ring-shaped region that captures any location whose distance from q' falls into the interval $[R - \tau, R + \tau]$. The best result is updated to be p_1. Next, point p_2 is retrieved (see Figure 11b). Since q is closer to p_2 than to p_1, the best result is updated to be p_2. The next point retrieved is p_3, as shown in Figure 11c. Since the termination condition is satisfied, the algorithm stops and returns p_2 as the result.

SQL Implementation.
The following SQL query, which is a variant of the queries presented earlier, enables incremental ring ranking using a conventional relational DBMS:

```
SELECT    id, x, y
```

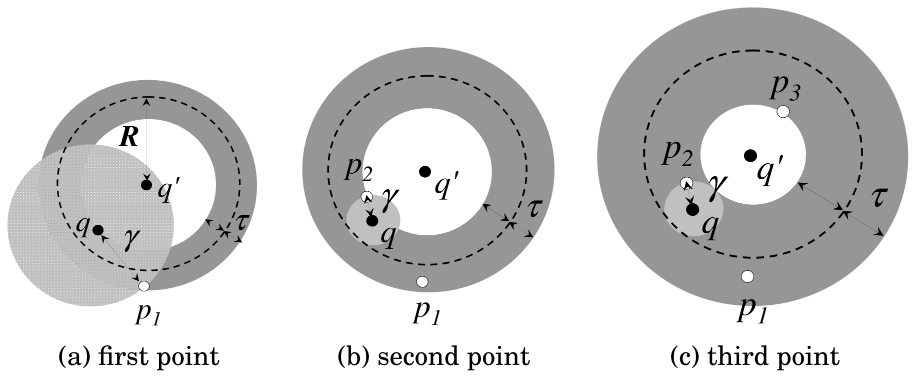


Fig. 11. Incremental Ring Ranking Example

```
FROM       P
ORDER BY   ABS( (x-_qx')·(x-_qx') + (y-_qy')·(y-_qy') - _R*_R )  ASC
```

Privacy Analysis.
We proceed to extend the privacy analysis of Section 4.2 for this modified version of SpaceTwist.

Let q_c be a *possible* user location, let m be the number of points received by the client, and let the points received (in their order of retrieval) be $p_1, p_2, \ldots, p_m$. The inferred privacy region Ψ is then defined by the set of all q_c satisfying the two inequalities given next.

As the algorithm did not terminate by the last point of the $(m-1)^{st}$ point, we obtain:

$$|dist(q_c, q') - R| + \overset{k}{\underset{1 \le i \le (m-1)}{\min}} dist(q_c, p_i) > dist(q', p_{(m-1)}) \, ,$$

where the middle term denotes the k^{th} smallest distance between the first $m-1$ points and q_c.

Since the algorithm did terminate by the last point of the m^{th} point, we derive:

$$|dist(q_c, q') - R| + \overset{k}{\underset{1 \le i \le m}{\min}} dist(q_c, p_i) \le dist(q', p_m)$$

6.3. Offering Privacy Guarantees by Delaying Termination

It is relevant to consider scenarios where the user specifies a privacy value threshold α and requires SpaceTwist to offer such a privacy value. This section develops a technique that provides such a privacy guarantee.

A simple heuristic is to select an anchor location q' such that $dist(q, q') = \alpha$. Experimental results (see Section 7) suggest that the measured privacy value Υ is usually above α. However, no theoretical result exists that guarantees that the measured privacy Υ is always above α.

Delayed Termination in SpaceTwist.
To offer the above guarantee, we propose to delay the termination of the SpaceTwist client algorithm until the measured privacy is above the required privacy value α. The pseudo code of this modified client is shown in Algorithm 4. In comparison to the original SpaceTwist client, it takes α as an additional parameter.

First, the set W_k is initialized to keep track of the best k objects found so far. Then it reuses the original SpaceTwist client functionality (Lines 2–11 of Algorithm 1) in order to retrieve the results and store them in the set W_k. In Line 3, the privacy region

Ψ is computed by using the points received. The computation of Ψ will be explained shortly. The algorithm then checks whether the measured privacy value $\Upsilon(q, \Psi)$ is below the required privacy value α. If so, it retrieves the next point from the server and recomputes the privacy region Ψ. This process is repeated until the required privacy is guaranteed, i.e., $\Upsilon(q, \Psi) \geq \alpha$. After that, the algorithm terminates the INN query at the server and reports W_k as the result to the user.

It is worth noticing that the algorithm returns the correct result to the user while guaranteeing that the measured privacy value is above the required α.

Algorithm 4 Delayed Termination SpaceTwist Client

```
   algorithm DelaySpaceTwistClient(Value $k$, Point $q$, Point $q'$, Required Privacy $\alpha$)
1: $W_k \leftarrow$ new max-heap of pairs $\langle p, dist(q,p) \rangle$;
2: apply Lines 2–11 of Algorithm 1;
3: compute the privacy region $\Psi$ by using all received points;
4: while $\Upsilon(q, \Psi) < \alpha$ do
5:     $p \leftarrow$ get the next point from the server;
6:     compute the privacy region $\Psi$ by using all received points;
7: terminate the INN query at the server;
8: return $W_k$;
```

Privacy Analysis.
Using the notation from Section 4.2, we consider the derivation of the privacy region Υ (occurs in Lines 3 and 6).

Let m be the number of received points at the moment when Υ is computed. Suppose that q_c is a possible user location. As the algorithm returns the correct result, the adversary learns that q_c must satisfy Inequality 4. On the other hand, since the adversary does not know the value of α, it cannot determine how many of those m points are the extra points retrieved in Line 5. Thus no additional inequalities can be used to further constrain the possible location q_c. Therefore, the privacy region Ψ is computed as the set of all locations that satisfy Equation 4.

6.4. Application of SpaceTwist to Road Networks

We proceed to apply SpaceTwist to the context where the locations of points are constrained to a road network.

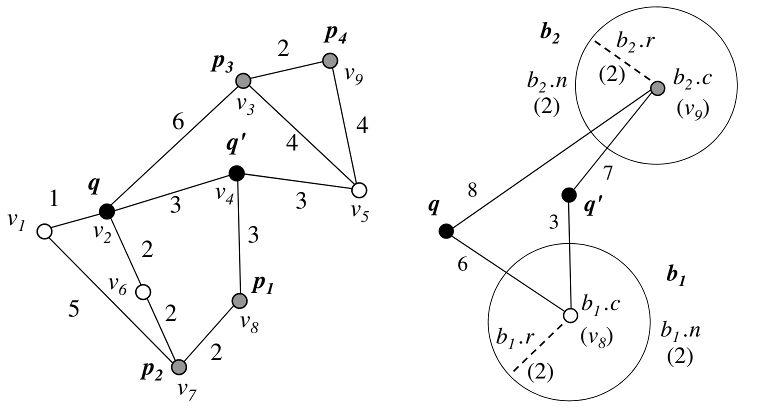
Graph-Based SpaceTwist.
We represent a spatial network by a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges (i.e., pairs of vertices from V). Each edge (v_a, v_b) is associated with a non-negative weight $w(v_a, v_b)$. We assume that every edge is bidirectional. Given any two vertices v_i and v_j, the *network distance* $dist_G(v_i, v_j)$ denotes the sum of the weights along the shortest path between v_i and v_j.

As in related work [Duckham and Kulik 2005a; 2005b], we assume for simplicity that data points can only be located at graph vertices. However, our solution is directly applicable to a road network model with data points located on edges [Papadias et al. 2003]. Figure 12a illustrates an example road network with eight vertices $(v_1, v_2, \ldots, v_8)$ and ten edges. For instance, the weight of the edge (v_2, v_4) is 3. The network distance $dist_G(v_4, v_7)$ is computed as $3 + 2 = 5$. The data points p_1, p_2, and p_3 are located on the vertices v_8, v_7, and v_3, respectively.

We assume that the server stores the road network G and the dataset P, and we assume that it supports the incremental network expansion (INE) algorithm [Papadias et al. 2003], which is an extension of Dijkstra's algorithm that incrementally retrieves nearest neighbors (of a given query point) with respect to the network distance. The client side is also required to keep a copy of the road network G so that it can compute network distances between points.

Algorithm 1 can be applied to the network space domain when replacing the Euclidean distance function $dist(\cdot)$ by the network distance function $dist_G(\cdot)$ and replacing the call to INN (in Line 5) by a call to INE. Since the network distance satisfies the triangular inequality, Lemma 4.1 also holds for the network space.

The main advantage of our algorithm over existing work [Ku et al. 2007; Li et al. 2008; Mouratidis and Yiu 2010; Wang and Liu 2009] is that our algorithm is easy to implement as it does not perform low-level operations (e.g., computing border intersections between the cloaked region and road segments).



(a) road network (b) network-based histogram

Fig. 12. Network-Based SpaceTwist Example

Example.
Figure 12a demonstrates the running steps of the network-based SpaceTwist algorithm for $k = 1$. The actual user location q is vertex v_2. Assume that the client chooses vertex v_4 to be the anchor q'. The client then requests the server to return points in ascending order of their network distances to q'.

First, the point p_1 (with $dist_G(q', p_1) = 3$) is received from the server. The algorithm updates τ to 3 and computes the network distance $dist_G(q, p_1) = 6$. The best result so far is p_1, and γ is set to 6. Next, point p_2 (with $dist_G(q', p_2) = 5$) is returned from the server. The value of τ is updated to 5. Since $dist_G(q, p_2) = 4$ is lower than γ, the algorithm sets the best result to p_2 and sets γ to 4. When point p_3 (with $dist_G(q', p_3) = 7$) is received, τ is updated to be 7. Now, the termination condition $\gamma + dist_G(q', q) \leq \tau$ is satisfied, so the algorithm terminates and returns p_2 as the result.

Privacy Analysis.
We assume that the adversary knows the road network being used by the user. Any valid location must fall into the road network. The privacy model of Sections 3.2 and 4.2 (including Equations 1, 2, 3, and 4) is directly applicable to the network-based SpaceTwist algorithm. It suffices to replace the Euclidean distance function $dist(\cdot)$ by the network distance function $dist_G(\cdot)$.

Communication Cost Estimation and Anchor Generation.
We then extend our techniques in Sections 5.4 and 5.5 for estimating the communication cost and generating the anchor for the network-based SpaceTwist. These issues

have not been studied in existing privacy solutions on road networks [Ku et al. 2007; Li et al. 2008; Mouratidis and Yiu 2010; Wang and Liu 2009].

In a spatial histogram, each bin has a rectangular extent, which cannot capture well the proximity of points based on network distances. Thus, we will adopt the *distance-based* representation [Ciaccia et al. 1997] for a bin in a network-based histogram. Specifically, each bin b_i is associated with its count of points $b_i.n$, its center $b_i.c$, and its radius $b_i.r$, where $b_i.r$ denotes the maximum network distance from $b_i.c$ to all data points in b_i. By Ciaccia et al. [1997], the minimum and maximum distances from a query point q to a bin b_i can be computed as follows:

$$mindist_G(q, b_i) = \max\{0, dist_G(q, b_i.c) - b_i.r\}$$

$$maxdist_G(q, b_i) = dist_G(q, b_i.c) + b_i.r$$

Figure 12b shows a histogram for the road network in Figure 12a. This histogram contains two bins b_1 and b_2. Bin b_1 has 2 points, its center as v_8, and its radius as 2. Bin b_2 has 2 points, its center as v_9, and its radius as 2. For example, the minimum and maximum distances from q to bin q_1 are $mindist_G(q, b_i) = 6 - 2 = 4$ and $maxdist_G(q, b_i) = 6 + 2 = 8$.

This new histogram renders the estimation techniques in Sections 5.4 and 5.5 applicable to road networks. This involves *cardinality estimation* and *anchor generation*. First, we briefly discuss how to estimate the count of data points within distance R of anchor q'. Let us consider how many points in a bin b_i that contribute to the estimated count. If $mindist_G(q, b_i) > R$ then no points in b_i can contribute to the estimated count. If $maxdist_G(q, b_i) \le R$ then all points in b_i contribute to the estimated count. Otherwise, we estimate the contribution of b_i to the estimated count as:

$$b_i.n \cdot \left(\frac{R - mindist_G(q, b_i)}{maxdist_G(q, b_i) - mindist_G(q, b_i)} \right)$$

Regarding the anchor generation in Algorithm 2, Line 1 is modified to choose a random network vertex as the anchor q', and Line 14 is modified to shift the location of the anchor q' such that its network distance from q becomes d'.

What remains is to discuss how to construct the bins in the above histogram. In order to obtain accurate estimates, we aim at minimizing the sum of the radii of the bins. This can be achieved by the M-tree [Ciaccia et al. 1997] which indexes data points based on a given metric (distance function). We set its distance metric to the network distance, and the node fanout to N/B, where N is the dataset size and B is the number of bins. We insert data points into the M-tree and use each resulting leaf node information as a bin.

Other Extensions.
All our extensions (i.e., granular search, ring-based ranking, and delayed termination) can be adapted to road networks.

7. EMPIRICAL EVALUATION

We evaluate five versions of SpaceTwist that employ different combinations of client-side and server-side algorithms.

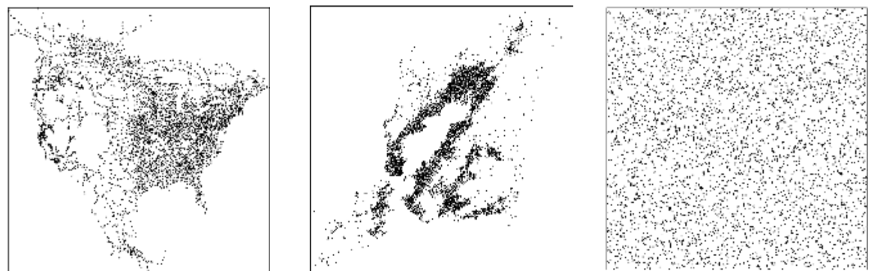
— The *Basic SpaceTwist* (BST) uses Algorithm 1 on the client and the incremental nearest neighbor algorithm on the server.
— The *Granular SpaceTwist* (GST) employs Algorithm 1 on the client, but uses Algorithm 3 on the server.

— The *Ring-Based SpaceTwist* (RST) corresponds to the proposal in Section 6.2, where modified client-side and server-side algorithms are used for the incremental ring ranking.
— The *Delayed Granular SpaceTwist* (DGST) is as GST, except that the client delays termination as described in Section 6.3.
— The *Network-Based SpaceTwist* (NST) applies the network-based version of Algorithm 1 on the client and the incremental network expansion algorithm on the server, as discussed in Section 6.4.

Section 7.2 studies a realistic scenario in which the client is a real mobile device and the server uses an SQL-based implementation. Section 7.3 examines the accuracy of the cost model (developed in Sections 5.4, 5.5, and 6.4). We then compare our solutions (BST, RST, GST) with existing client-server approaches in Section 7.4. Section 7.5 studies the performance of GST when varying the settings of pertinent parameters. Then, Section 7.6 investigates the effect of the delayed termination condition on the performance of DGST. Section 7.7 evaluates the performance of NST on real road networks. Finally, Section 7.8 summarizes the findings.

7.1. Settings

For the experiments in Euclidean space, we use synthetic datasets (UI) that are randomly uniformly generated, and we also use two real datasets: NA[5] (North America) with 175,813 points, and SF [Brinkhoff 2002] (San Francisco) with 174,956 points. These datasets are visualized in Figure 13. For the experiments in road networks, we use two real road networks: NA with 175,813 vertices and 179,179 edges; and SF with 175,813 vertices and 223,001 edges. The *data density* of a network denotes the fraction of graph vertices with data points. For each tested case, we randomly generate data points such that the data density equals a given value. The default data density is 0.01.



(a) North America: NA (b) San Francisco: SF (c) Uniform data: UI

Fig. 13. Datasets for Experiments

The coordinates of points in each dataset are normalized to the square 2D space with side length 10,000 meters. A 2D data point takes 20 bytes, as its identifer takes 4 bytes and each coordinate takes 8 bytes. Except in the study of the SQL implementation (in Section 7.2), we employ an LBS server, which indexes each dataset by an R-tree with a 4K byte page size.

Table II summarizes the parameters used in the experiments along with their settings, with default values in bold. In each experiment, we use a workload of 100 ran-

[5]Digital Chart of the World, `www.maproom.psu.edu/dcw/`.

Table II. Parameter Values

Parameter	Values	Applicability
Error bound (meter), ϵ	0, 50, 100, **200**, 500, 1000	GST, DGST
Anchor distance (meter), $dist(q, q')$	50, 100, **200**, 500, 1000	all
Number of required results, k	1, 2, **4**, 8, 16	all
Data size (million), N	0.1, 0.2, **0.5**, 1, 2	all

domly generated query points and measure the average value of the following performance metrics:

— Communication cost, in numbers of points received by the client.
— Result error, defined as the result kNN distance minus the actual kNN distance.
— Privacy value of the inferred privacy region (Equation 1).

As discussed in Section 4.2, the privacy value needs to be computed by using a technology-dependent interval bound $[\beta^-, \beta^+]$ that captures the number of additional/unnecessary points sent by the server. We determine these values in the next section.

7.2. Mobile Devices and SQL Implementation

In this experiment, we use a conventional DBMS server (SQL Server 2005, Intel Xeon 4-core CPU 2.33GHz) and two mobile client devices: an Asus P535 with Wi-Fi and GPRS connectivity and an HTC Diamond that supports 3G. The end-user time is measured as the time between the `Start` and `Stop` messages sent by the client. It includes the client's processing time as the `Stop` message can only be sent after the client has examined the retrieved points and determined that no more points are needed. We only report the performance of the Basic SpaceTwist (BST) on the default UI dataset (with 500,000 points). When performing this experiment on real datasets (NA and SF), we obtain similar results. It is worth noticing that existing spatial anonymization, obfuscation, and private retrieval methods cannot be readily expressed as a single SQL statement on the dataset.

Table III shows the average end-user time of BST for different connections, as a function of the anchor distance $dist(q, q')$. The average communication cost (i.e., number of points received by client, and its equivalence in Kbytes) of each case is also shown for reference. Note that all these end-user times are far smaller than those reported by the naive method in the introduction. With either Wi-Fi or 3G, the client is able to afford high privacy value (e.g., $dist(q, q') = 1000$) with only 6 seconds. Even with the slow GPRS connection, the client can enjoy medium privacy (e.g., $dist(q, q') = 200$) in reasonable time.

Table III. Average End-User Time of BST vs. Anchor Distance $dist(q, q')$; UI data

Anchor distance (meter)	*Average number of points received*	*Equivalent Kbytes*	*Time (s)*		
			Wi-Fi	**GPRS**	**3G**
50	58.67	1.14	2.16	5.05	2.50
100	186.57	3.64	2.22	5.06	2.55
200	677.89	13.24	2.29	6.51	2.62
500	3748.05	73.20	2.88	18.50	3.27
1000	14593.03	285.02	3.45	60.21	6.01

Figure 14 plots the end-user time of BST per instance, for different connection types, while fixing $dist(q, q') = 200$. Observe that these instances do not form a straight line in the figure. Instead, they fall into a cluster, whose width is 1 s and whose length is $(750 - 625) = 125$ points. Thus, we fix the value of $\beta^+ - \beta^-$ to 125, when measuring the privacy values in subsequent experiments.

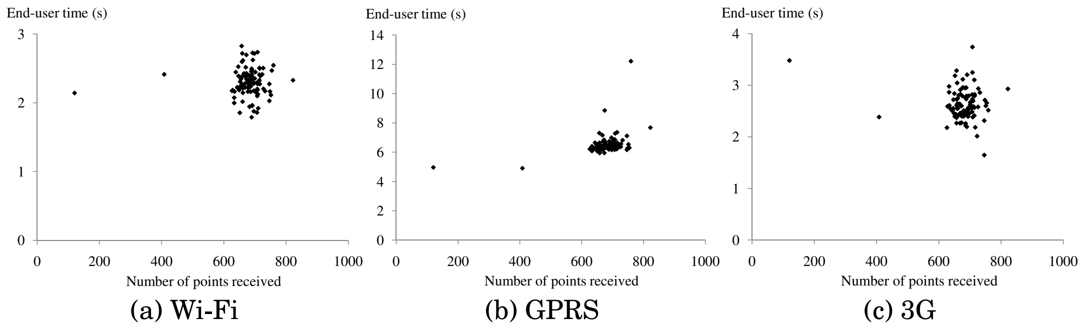


Fig. 14. End-User Time of BST per Instance; UI data, Anchor Distance=200m

7.3. Communication Cost Model Accuracy

We investigate the accuracy of the communication cost model developed in Sections 5.4 and 5.5. An anchor q' is selected such that its estimated cost $cost_{est}$ equals a given communication cost budget. Let $cost_{act}$ be the actual cost of running BST with such an anchor. The relative cost error is then measured as: $\frac{|cost_{est} - cost_{act}|}{cost_{act}}$. By default, k is fixed at 4 and, the cost budget is set to 2000 points, in the following experiments. We employ a spatial histogram with 1024 bins, whose communication cost overhead is only $1024 \cdot (4 + 4 \cdot 4)/1024 = 20$ Kbytes, where each bin consists of a count (an 4-byte integer) and its discretized bounding box (four 4-byte integers). It is possible to apply existing histogram compression techniques to further shrink the histogram size, by only sacrificing its quality a little.

Figure 15a shows the relative cost error for different datasets, with respect to a given communication cost budget. Recall from Algorithm 2 that the estimated supply space radius $\widehat{\tau}$ equals the sum of the anchor distance $dist(q, q')$ and the estimated kNN distance $\widehat{\gamma}$. When the budget is high, a large $dist(q, q')$ is selected, and it contributes to a large fraction of $\widehat{\tau}$. Even with an imperfect estimation of $\widehat{\gamma}$, its impact on $\widehat{\tau}$ remains small, keeping the overall estimation error relatively low. Following the intuition, the estimation error on UI data is smaller than that on the real datasets (NA and SF). When a sufficient budget is used (e.g., 2000 points), our estimation method yields a reasonable estimation error (0.3–0.6).

Figure 15b plots the relative cost error on different datasets, as a function of k. When k increases, the estimated kNN distance $\widehat{\gamma}$ becomes more accurate, thus reducing the overall estimation error of our cost model.

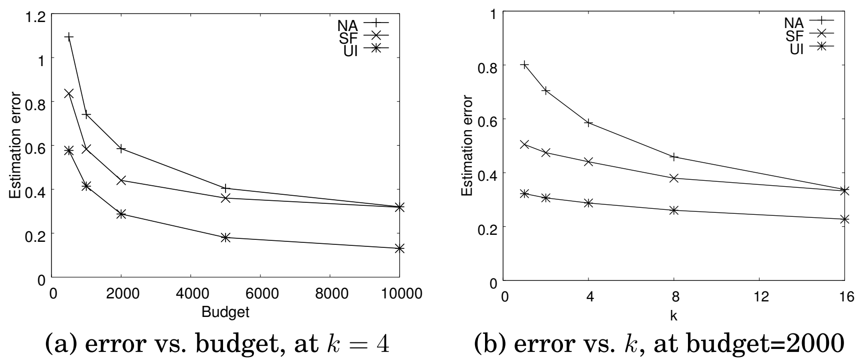
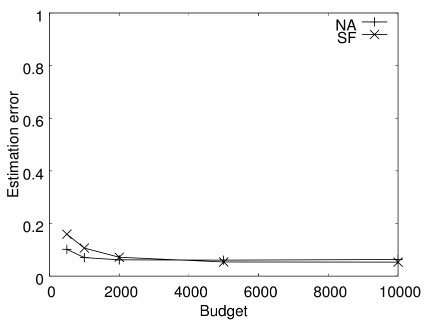


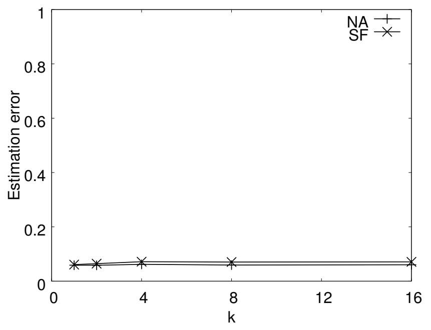
Fig. 15. Relative Error of Communication Cost, with an 1024-bin Spatial Histogram

We proceed to evaluate the accuracy of our network-based anchor generator in Section 6.4. We also use the aforementioned measurement for the relative cost error, except that $cost_{act}$ now denotes the actual cost of running NST (Network-Based SpaceTwist) with the generated anchor. We also employ a network histogram with 1024 bins, whose communication cost overhead is $1024 \cdot (4 + 8 + 4)/1024 = 16$ Kbytes, where each bin consists of a center node ID (4-byte integer), a radius (8-byte double), and a count (4-byte integer).

Figure 16a shows the relative cost error on two real road networks when varying the communication cost budget. It follows the decreasing trend from Figure 15a. The inherent dimensionality of a road network space lies between 1 (linear) and 2 (planar). When compared to Euclidean space, a data point in a road network is expected to have fewer 'neighbors', and thus the estimation error is lower. Figure 16b plots the relative cost error on two real road networks, with respect to k. The error remains very low and is insensitive to the value of k.



(a) error vs. budget, at $k = 4$



(b) error vs. k, at budget=2000

Fig. 16. Relative Error of Communication Cost, with an 1024-bin Network Histogram

7.4. Comparison with Existing Client-Server Approaches

In keeping with the simple client-server architecture (assumed in our problem setting), we disregard techniques that require trusted third-party components or peer-to-peer functionality.

For comparison purposes, we implement a prototype client-based cloaking (i.e., obfuscation) technique, called CLK, that generates the cloaked region as a (randomly generated) square region that contains the exact user location q. The region has an extent of $2 \cdot dist(q, q')$, making it comparable to the inferred privacy region of GST. The value $dist(q, q')$ roughly reflects the privacy value, as we will see shortly in Section 7.5. The query processing algorithm of Hu and Lee [2006] (used in Kalnis et al. [2007]) is applied on the server to evaluate the cloaked query such that the minimal candidate result set is reported.

Since CLK provides exact results, we include our exact solutions BST and RST in this study. The ring radius R of RST is simply set to the anchor distance $dist(q, q')$. The result accuracy of GST is guaranteed by a user-specified error bound ($\epsilon = 200$ by default).

Table IV shows the communication cost as a function of $dist(q, q')$ for different datasets. The cost rises when $dist(q, q')$ increases. Since RST is designed towards reducing accesses to the "center region" surrounding the anchor, it outperforms BST in all cases. In addition, RST incurs lower cost than CLK when $dist(q, q')$ is sufficiently large. Clearly, GST has the lowest communication cost because it is able to discard uninformative points that stay close to previously reported points.

From this experiment, we conclude that the cost of CLK does not scale well with the extent of the cloaked region.

Table IV. Communication Cost vs. $dist(q, q')$

	NA				*SF*				*UI, N=0.5M*			
$dist(q, q')$	**BST**	**RST**	**GST**	**CLK**	**BST**	**RST**	**GST**	**CLK**	**BST**	**RST**	**GST**	**CLK**
50	80.4	72.3	14.9	33.9	64.1	56.7	14.8	24.0	67.5	49.0	18.4	63.9
100	218.4	175.4	26.1	103.0	181.4	146.9	26.1	74.5	207.0	95.5	33.6	215.6
200	647.0	438.3	51.7	361.6	513.9	362.8	52.1	281.6	708.9	188.0	68.1	802.4
500	3064.0	1598.2	170.6	2106.5	2317.4	1391.7	162.7	1774.3	3916.0	450.0	238.3	4783.8
1000	9284.8	4179.6	466.6	8234.2	8268.6	4506.9	438.7	6571.2	14142.8	821.6	709.5	18057.8

Recall from Section 2.3 that private retrieval methods do not leverage spatial database technology so they do not satisfy our high deployability requirement. Nevertheless, we still compare GST with the following approximate solutions: (i) SHB [Khoshgozaran and Shahabi 2007], which returns the k nearest neighbors along a Hilbert curve, (ii) DHB [Khoshgozaran and Shahabi 2007], which performs search along two orthogonal Hilbert curves, and (iii) APX [Ghinita et al. 2008], a two-dimensional approximate method that returns all points of a $\mathbb{K}$-d tree [Bentley 1975] leaf node whose extent covers the query point. In accordance with existing studies, we fix the level of the Hilbert curves used at 12 for both SHB and DHB [Khoshgozaran and Shahabi 2007], and we set the number of leaf nodes to $\sqrt{N}$ for APX [Ghinita et al. 2008]. A previous theoretical study [Indyk and Woodruff 2006] does not offer implementation details and is not covered here. The exact solution of Ghinita et al. [2008] is also not considered as it utilizes a Voronoi diagram, which cannot be used to answer kNN queries.

Table V compares the result error of SHB, DHB, APX, and GST for different values of k. For the real-world datasets (NA and SF), SHB computes results with poor accuracy because a Hilbert curve only preserves spatial proximity approximately. Since DHB employs two Hilbert curves, it is more accurate than SHB. Note that APX is even more accurate because it preserves data locality better than Hilbert curves do. However, the average error of APX is not small because it can suffer from inaccuracies in some cases where the query point is located close to the border of multiple $\mathbb{K}$-d tree leaf nodes. Note that GST benefits from skew in the data to achieve the best accuracies. This is so because data points in the same grid cell (as illustrated in Figure 9b) are likely to fall in the same cluster and the distances between them are significantly lower than the worst case distance bound (i.e., the diagonal grid cell distance).

For uniform data (UI), the Hilbert transformation approach (SHB and DHB) and the two-dimensional approximate method APX are quite accurate. Nevertheless, the accuracy of GST remains acceptable, being much better than the specified error bound ($\epsilon = 200$).

Table V. Result Error Versus k

	NA				*SF*				*UI, N=0.5M*			
k	**GST**	**SHB**	**DHB**	**APX**	**GST**	**SHB**	**DHB**	**APX**	**GST**	**SHB**	**DHB**	**APX**
1	12.9	316.2	99.4	41.9	11.8	218.0	109.3	42.8	43.6	9.2	4.9	0.0
2	13.6	328.5	122.0	42.4	11.3	242.9	115.8	49.9	44.1	11.0	5.6	0.5
4	13.7	359.3	139.7	42.3	9.6	343.7	182.8	49.8	43.2	13.1	5.6	0.6
8	14.3	381.9	161.3	43.0	8.0	376.7	187.6	51.0	41.5	19.5	7.2	1.2
16	12.5	400.4	178.8	45.1	8.2	457.1	189.7	57.9	35.4	27.1	10.1	2.6

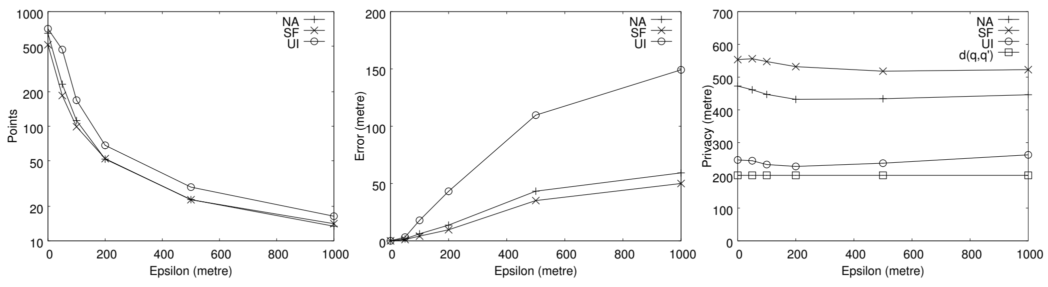
In summary, GST is robust and achieves stable result errors across different data distributions. Existing approximate solutions (SHB, DHB, and APX) do not offer result error guarantees as provided by GST. Also, their result errors are much higher than those of GST for real datasets.

7.5. Performance Study of GST

We proceed to investigate the scalability of GST with respect to different parameters while using the two real datasets as well as the UI dataset.

Figure 17 depicts the performance of GST when varying the error bound ϵ. As a reference for comparison, the curve for the anchor distance $dist(q, q')$ is included in Figure 17c. As ϵ increases, each grid cell has a larger extent, and fewer points are retrieved, which yields a lower communication cost, but a larger result error. Since the real datasets are skewed, the average error for these is much smaller than the specified error bound ϵ. At $\epsilon = 0$, granular search is not applied, and exact results are reported. Even for this case, the communication cost and privacy value are both acceptable.

Observe that GST indeed achieves both low communication cost and low result error for a broad range of ϵ values (between 100 and 1000) when applied to real data. At $\epsilon = 100$, the communication cost is only 200 points at most. For the other end (i.e., $\epsilon = 1000$), the measured error is acceptably low and stays within 25% of the bound ϵ for real datasets.



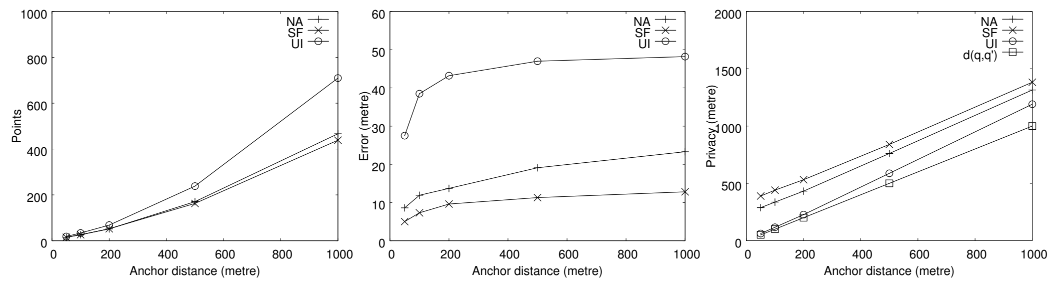
(a) communication cost (b) result error (c) privacy value

Fig. 17. Performance of GST Vs. Error Bound ϵ

Figure 18 shows the performance of GST as a function of the anchor distance $dist(q, q')$. The communication cost and result error increase when $dist(q, q')$ increases. However, even for large $dist(q, q')$, the communication cost and result error are quite low. Note also that the location privacy afforded by GST is greater than the anchor distance $dist(q, q')$. It is worth noticing that the more the skew, the lower the result error and the higher the privacy value become.

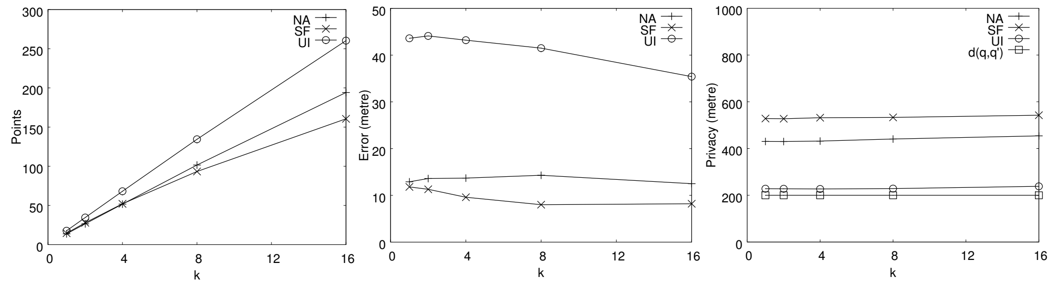
Figure 19 shows the performance of GST when varying the number of required results k. The communication cost is proportional to k, and it remains low for typical values of k. Both the result error and the privacy value are fairly insensitive to k, but benefit from skew in the data. For real datasets, the privacy value is much larger than the specified anchor distance.

We end the study of GST by varying the dataset size N using synthetic UI datasets. Figure 20 plots the results. Since the error bound ϵ is fixed, the communication cost, result error, and privacy are insensitive to N. Thus, GST scales well with the dataset size.



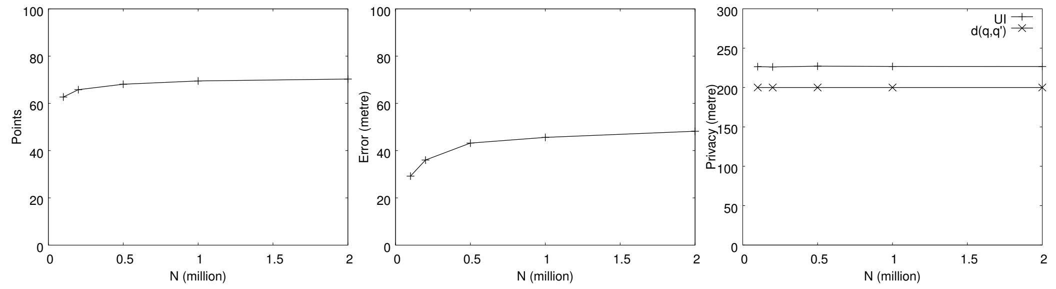
(a) communication cost (b) result error (c) privacy value

Fig. 18. Performance of GST Vs. Anchor Distance $dist(q, q')$



(a) communication cost (b) result error (c) privacy value

Fig. 19. Performance of GST Vs. Number of Required Results k



(a) communication cost (b) result error (c) privacy value

Fig. 20. Performance of GST Vs. Data Size N, on UI Datasets

7.6. Performance of the Delayed Termination Strategy

We proceed to study the effect of the required privacy value α on the performance of DGST. In this experiment, we vary the value of α while fixing the other parameters (including the anchor distance) at their default values. The measured result error of DGST is independent of α so it is not reported.

Figure 21a plots the measured privacy value of DGST with respect to the required privacy value α. The requirement curve for α is included in the figure for comparison purposes. Observe that the measured privacy values on all datasets are now guaranteed to exceed α. The privacy on all datasets stay very close to α.

Figure 21b shows the communication cost of DGST as a function of α. When α increases, a larger number of points need to be retrieved in order to guarantee that the privacy value exceeds α. Even for the cases of high privacy requirements (e.g., $\alpha = 2000$), the number of received points on any dataset remains below 5000.

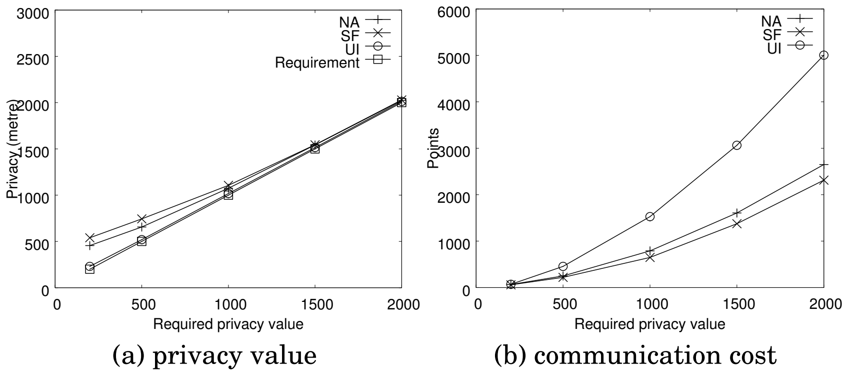


Fig. 21. Performance of DGST Vs. Required Privacy Value α

7.7. Performance of the Network-Based SpaceTwist

In the last two experiments, we study the performance of NST. We use two real road networks (NA and SF) and the experimental setting described in Section 7.1. The default value of the network anchor distance $dist_G(q', q)$ is fixed at 200.

Figure 22 shows the communication cost and privacy value of NST for various data densities. Since the anchor distance is fixed, many data points are received from a network with high data density. Recall from Section 4.2 that, for the given values q' and k, the number of distinct inferred privacy regions is N, which is directly proportional to the number of data points. When the data density increases, the value N increases, implying that the "thickness" of each individual (irregular ring-shaped) privacy region shrinks. The combined effect is that the extent of the overall privacy region is insensitive to the data density.

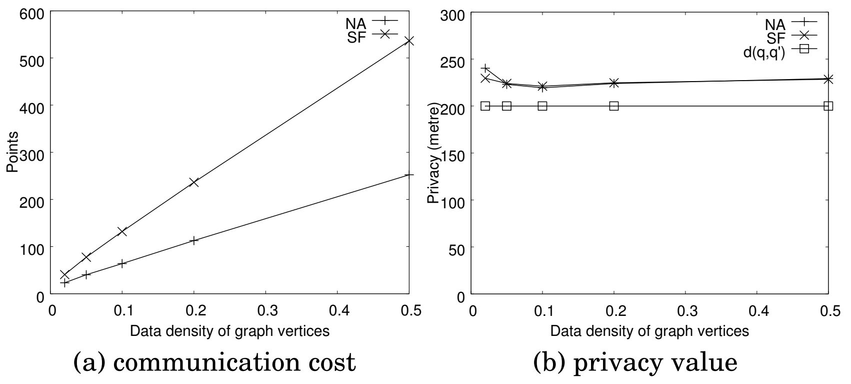
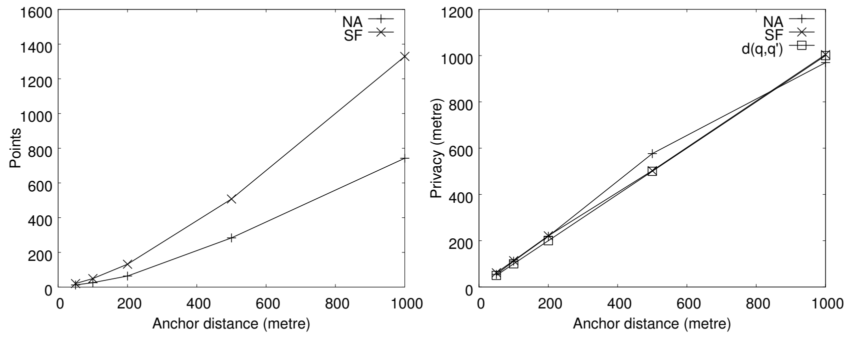


Fig. 22. Performance of NST Vs. Data Density of Graph Vertices

Figure 23 plots the communication cost and privacy value of NST when varying the network anchor distance $dist_G(q, q')$. When the anchor distance increases, the communication cost rises slightly in a super-linear manner and the privacy value increases linearly. Observe that the cost remains small even at a large anchor distance.



(a) communication cost (b) privacy value

Fig. 23. Performance of NST Vs. Network Anchor Distance $dist_G(q, q')$

7.8. Summary of Experimental Results

We proceed to summarize our findings from the experimental results.

First, our proposal is readily deployable using conventional DBMS servers. Our SQL implementation of SpaceTwist is able to achieve high privacy in reasonable end-user time.

Second, our cost model is acceptably accurate, allowing the client to choose an appropriate anchor point based on the communication cost budget. The relative cost error of our estimation ranges from 0.3 to 0.6 in most of the cases studied.

Third, our methods perform better than existing solutions in terms of communication cost and result error. For instance, our GST incurs much lower communication cost than the cloaking method CLK. Our RST has lower cost than CLK when $dist(q, q')$ is sufficiently large. For the real datasets, GST achieves much better result error than private retrieval methods (SHB, DHB, and APX).

Fourth, our GST method performs well for wide ranges of settings of parameters such as the error bound ϵ, the anchor distance $dist(q, q')$, the number of required results k, and the data size N.

For the GST method, the user needs to specify the result error bound ϵ and an anchor location q'. It is desirable to choose ϵ based on the data density and the value of k. We recommend to set $\epsilon = \sqrt{\frac{k}{(0.01)N}}$. This choice saves substantial communication cost (i.e., the worst-case cost is 1% of data points) and yet ϵ becomes adequately small for a large dataset. The choice of q' can be determined automatically by Algorithm 2, based on a given cost budget. Alternatively, q' can be selected such that anchor distance $dist(q, q')$ equals a given privacy value. Experimental results show that the measured privacy value of GST is usually above the value of $dist(q, q')$.

To guarantee that the measured privacy is always above the value of $dist(q, q')$, the DGST method applies a delayed termination strategy.

Last but not least, in the road network environment, our network-based SpaceTwist is readily deployable as it directly applies the network-based incremental nearest neighbor algorithm, without caring low-level access operations on the road network.

8. CONCLUSION AND RESEARCH DIRECTIONS

This paper concerns the efficient support for location privacy of users of location-based service. Existing location privacy solutions either incur high server-side loads, require specialized server techniques, or produce results without worst-case guarantees on the accuracy bounds of the query results.

Motivated by this, the paper proposes a novel and effective framework, called SpaceTwist, that consists of a client-side algorithm and a server-side granular search technique that supports user-defined (relaxed) query accuracies. SpaceTwist offers fine-grained support for managing the trade-offs among location privacy, query performance, and query accuracy. Empirical studies with real-world datasets demonstrate that SpaceTwist is capable of providing high degrees of location privacy as well as very accurate results at low communication cost.

Furthermore, SpaceTwist is the first location privacy solution whose server-side functionality can be expressed as a single SQL query on the queried dataset. We also contribute extensions of SpaceTwist that provide specialized services for other application scenarios, including a delayed termination strategy for enhancing the privacy protection, an incremental ranking technique for reducing the communication cost, and a solutions that renders SpaceTwist applicable to spatial networks.

A promising direction is to generalize SpaceTwist for private ranking queries over public data (of generic types). The goal here is to hide the user's query interest (i.e., a conceptual 'location' defined in the data space). As an example scenario, a doctor wishes to search a public database of medical MRI images for an image similar to a patient's MRI image; however, the doctor is not allowed to expose the patient's image to third parties. Interesting aspects of the privacy model and the choice of anchor object need to be studied in this setting.

ACKNOWLEDGMENTS

Man Lung Yiu was supported by the grant PolyU 5333/10E from Hong Kong RGC. Christian S. Jensen is an Adjunct Professor at University of Agder, Norway. Jesper Møller was supported by the Danish Natural Science Research Council, grants 272-06-0442 and 09-072331, "Point process modelling and statistical inference", and by the Centre for Stochastic Geometry and Advanced Bioimaging, funded by a grant from the Villum Foundation.

REFERENCES

ACHARYA, S., POOSALA, V., AND RAMASWAMY, S. 1999. Selectivity Estimation in Spatial Databases. In *SIGMOD*. 13–24.

ARDAGNA, C. A., CREMONINI, M., DAMIANI, E., DI VIMERCATI, S. D. C., AND SAMARATI, P. 2007. Location Privacy Protection Through Obfuscation-Based Techniques. In *DBSec*. 47–60.

BAMBA, B., LIU, L., PESTI, P., AND WANG, T. 2008. Supporting Anonymous Location Queries in Mobile Environments with Privacygrid. In *WWW*. 237–246.

BARTOLINI, I., CIACCIA, P., AND PATELLA, M. 2006. SaLSa: Computing the Skyline without Scanning the Whole Sky. In *CIKM*. 405–414.

BENTLEY, J. L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM 18,* 9, 509–517.

BETTINI, C., MASCETTI, S., WANG, X. S., AND JAJODIA, S. 2007. Anonymity in Location-Based Services: Towards a General Framework. In *MDM*. 69–76.

BRINKHOFF, T. 2002. A Framework for Generating Network-Based Moving Objects. *GeoInformatica 6,* 2, 153–180.

CHENG, R., ZHANG, Y., BERTINO, E., AND PRABHAKAR, S. 2006. Preserving User Location Privacy in Mobile Data Management Infrastructures. In *Privacy Enhancing Technology Workshop*. 393–412.

CHOW, C.-Y. AND MOKBEL, M. F. 2007. Enabling Private Continuous Queries For Revealed User Locations. In *SSTD*. 258–275.

CHOW, C.-Y., MOKBEL, M. F., AND LIU, X. 2006. A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services. In *ACM GIS*. 171–178.

CIACCIA, P., PATELLA, M., AND ZEZULA, P. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*. 426–435.

DUCKHAM, M. AND KULIK, L. 2005a. A Formal Model of Obfuscation and Negotiation for Location Privacy. In *PERVASIVE*. 152–170.

DUCKHAM, M. AND KULIK, L. 2005b. Simulation of Obfuscation and Negotiation for Location Privacy. In *COSIT*. 31–48.

GEDIK, B. AND LIU, L. 2005. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *ICDCS*. 620–629.

GHINITA, G., KALNIS, P., KHOSHGOZARAN, A., SHAHABI, C., AND TAN, K.-L. 2008. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD*. 121–132.

GHINITA, G., KALNIS, P., AND SKIADOPOULOS, S. 2007a. MobiHide: A Mobile Peer-to-Peer System for Anonymous Location-Based Queries. In *SSTD*. 758–769.

GHINITA, G., KALNIS, P., AND SKIADOPOULOS, S. 2007b. PRIVÉ: Anonymous Location-Based Queries in Distributed Mobile Systems. In *WWW*. 371–380.

GRAVANO, L., IPEIROTIS, P. G., JAGADISH, H. V., KOUDAS, N., MUTHUKRISHNAN, S., AND SRIVASTAVA, D. 2001. Approximate String Joins in a Database (Almost) for Free. In *VLDB*. 491–500.

GRUTESER, M. AND GRUNWALD, D. 2003. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *USENIX MobiSys*. 31–42.

GRUTESER, M. AND LIU, X. 2004. Protecting Privacy in Continuous Location-Tracking Applications. *IEEE Security & Privacy 2,* 2, 28–34.

GUTTMAN, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

HJALTASON, G. R. AND SAMET, H. 1999. Distance Browsing in Spatial Databases. *TODS 24(2)*, 265–318.

HU, H. AND LEE, D. L. 2006. Range Nearest-Neighbor Query. *IEEE TKDE 18(1)*, 78–91.

ILYAS, I. F., AREF, W. G., ELMAGARMID, A. K., ELMONGUI, H. G., SHAH, R., AND VITTER, J. S. 2006. Adaptive Rank-aware Query Optimization in Relational Databases. *TODS 31,* 4, 1257–1304.

ILYAS, I. F., SHAH, R., AREF, W. G., VITTER, J. S., AND ELMAGARMID, A. K. 2004. Rank-aware Query Optimization. In *SIGMOD*. 203–214.

INDYK, P. AND WOODRUFF, D. 2006. Polylogarithmic Private Approximations and Efficient Matching. In *Theory of Cryptography Conference*. 245–264.

JAGADISH, H. V., OOI, B. C., TAN, K.-L., YU, C., AND ZHANG, R. 2005. iDistance: An Adaptive B$^+$-tree Based Indexing Method for Nearest Neighbor Search. *TODS 30,* 2, 364–397.

JENSEN, C. S., LU, H., AND YIU, M. L. 2009. Location Privacy Techniques in Client-Server Architectures. In *Privacy in Location-Based Applications*. 31–58.

KALNIS, P., GHINITA, G., MOURATIDIS, K., AND PAPADIAS, D. 2007. Preventing Location-Based Identity Inference in Anonymous Spatial Queries. *IEEE TKDE 19,* 12, 1719–1733.

KHOSHGOZARAN, A. AND SHAHABI, C. 2007. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*. 239–257.

KIDO, H., YANAGISAWA, Y., AND SATOH, T. 2005. An Anonymous Communication Technique using Dummies for Location-based Services. In *IEEE International Conference on Pervasive Services (ICPS)*. 1248.

KINGMAN, J. F. C. 1993. *Poisson Processes*. Oxford University Press.

KU, W.-S., ZIMMERMANN, R., PENG, W.-C., AND SHROFF, S. 2007. Privacy Protected Query Processing on Spatial Networks. In *ICDE Workshops*. 215–220.

LI, F., HADJIELEFTHERIOU, M., KOLLIOS, G., AND REYZIN, L. 2006. Dynamic Authenticated Index Structures for Outsourced Databases. In *SIGMOD*. 121–132.

LI, P.-Y., PENG, W.-C., WANG, T.-W., KU, W.-S., XU, J., AND HAMILTON, J. A. 2008. A Cloaking Algorithm Based on Spatial Networks for Location Privacy. In *SUTC*. 90–97.

LU, H., JENSEN, C. S., AND YIU, M. L. 2008. PAD: Privacy-area Aware, Dummy-based Location Privacy in Mobile Services. In *MobiDE*. 16–23.

MOKBEL, M. F., CHOW, C.-Y., AND AREF, W. G. 2006. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*. 763–774.

MØLLER, J. AND WAAGEPETERSEN, R. P. 2004. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, Boca Raton.

MØLLER, J. AND YIU, M. L. 2010. Probabilistic Results for a Mobile Service Scenario. *Research Report R-2010-03, Department of Mathematical Sciences, Aalborg University. (Revised version to appear in Advances in Applied Probability.)*.

MOURATIDIS, K. AND YIU, M. L. 2010. Anonymous Query Processing in Road Networks. *IEEE TKDE 22,* 1, 2–15.

OKABE, A., BOOTS, B., SUGIHARA, K., AND CHIU, S. 2000. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* second Ed. Wiley.

PANG, H., JAIN, A., RAMAMRITHAM, K., AND TAN, K.-L. 2005. Verifying Completeness of Relational Query Results in Data Publishing. In *SIGMOD*.

PAPADIAS, D., ZHANG, J., MAMOULIS, N., AND TAO, Y. 2003. Query Processing in Spatial Network Databases. In *VLDB*. 802–813.

PAPADOPOULOS, S., YANG, Y., BAKIRAS, S., AND PAPADIAS, D. 2009. Continuous Spatial Authentication. In *SSTD*. 62–79.

QUINE, M. P. AND WATSON, D. F. 1984. Radial Generation of n-Dimensional Poisson Processes. *Journal of Applied Probability 21*, 548–557.

ROUSSOPOULOS, N., KELLEY, S., AND VINCENT, F. 1995. Nearest Neighbor Queries. In *SIGMOD*. 71–79.

SWEENEY, L. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems 10(5)*, 557–570.

TAO, Y., ZHANG, J., PAPADIAS, D., AND MAMOULIS, N. 2004. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *IEEE TKDE 16,* 10, 1169–1184.

WANG, T. AND LIU, L. 2009. Privacy-Aware Mobile Services over Road Networks. *PVLDB 2,* 1, 1042–1053.

XIN, D., HAN, J., AND CHANG, K. C.-C. 2007. Progressive and Selective Merge: Computing Top-k with Ad-hoc Ranking Functions. In *SIGMOD*. 103–114.

XIN, D., HAN, J., CHENG, H., AND LI, X. 2006. Answering Top-k Queries with Multi-Dimensional Selections: The Ranking Cube Approach. In *VLDB*. 463–475.

XU, J., TANG, X., HU, H., AND DU, J. 2010. Privacy-Conscious Location-Based Queries in Mobile Environments. *IEEE TPDS 21,* 3, 313–326.

YANG, Y., PAPADOPOULOS, S., PAPADIAS, D., AND KOLLIOS, G. 2009. Authenticated Indexing for Outsourced Spatial Databases. *VLDB J. 18,* 3, 631–648.

YAO, B., LI, F., AND KUMAR, P. 2010. K Nearest Neighbor Queries and kNN-Joins in Large Relational Databases (Almost) for Free. In *ICDE*. 4–15.

YIU, M. L., JENSEN, C. S., HUANG, X., AND LU, H. 2008. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*. 366–375.

ZOU, L. AND CHEN, L. 2008. Dominant Graph: An Efficient Indexing Structure to Answer Top-K Queries. In *ICDE*. 536–545.

A. APPENDIX—TRANSFERRING RESULTS FROM THE SERVER TO THE CLIENT

In both the LBS and DBMS implementations discussed above, the server needs to return incremental NNs to the client by using a communication channel. For many programming languages (e.g., C++ and Java), high-level APIs are available that enable the server to send data points to the client via a TCP/IP socket interface. Low-level issues, such as TCP/IP packet sizes and the segmentation of data points into packets, are handled transparently by such APIs.

We consider two means by which the server can send incremental NNs to the client. Both are built on top of TCP/IP socket APIs, so they are guaranteed to provide reliable communication. The client may issue the following messages to the server: `Start`, `GetNext`, and `Stop`.

We first consider a straightforward method called *Blocking*. The client first sends a `Start` message to the server to initiate the process. Upon receiving a `GetNext` message from the client, the server sends the next NN point to the client. When the client decides to terminate the process, it sends a `Stop` message to the server. The major disadvantage of Blocking is that the server needs to wait for `GetNext` messages, and the number of round trips equals the number of points sent.

To eliminate the waiting time, we consider a method called *Non-blocking*. The client first sends a `Start` message to the server to initiate the process. Then, the server sends incremental NNs to the client iteratively (in a streaming manner), without waiting for `GetNext` messages. The client employs an event trigger that invokes necessary processing whenever a new data point arrives via the stream. When the client decides to terminate the process, it sends a `Stop` message to the server. Some unnecessary data points are likely to be sent from the server to the client during the time it takes for the eventual `Stop` message to be transmitted from the client to the server. Specifically, we denote the number of such unnecessary data points by $\beta^* = m'' - m$, where m denotes the number of points received by the client (when it sends the Stop message), and m'' denotes the total number of points sent by the server.

We compare the efficiency of Blocking and Non-blocking experimentally. Specifically, we use a mobile device (an Asus P535) with two popular wireless connections (Wi-Fi and GPRS), and another mobile device (an HTC Diamond) with a 3G connection. We measure the end-user time as the time between the Start and the Stop messages sent by the client. It includes the client's processing time as the Stop message can only be sent after the client has examined the retrieved points and determined that no more points are needed. For the Non-blocking method, we also measure the number of data points sent by the server. Figure 24a shows the end-user time of the methods with respect to the number of points received by the client. Note that both axes are shown in log scale, whereas the label suffixes B and the NB refer to Blocking and Non-blocking respectively. Clearly, Non-blocking outperforms Blocking for all connection

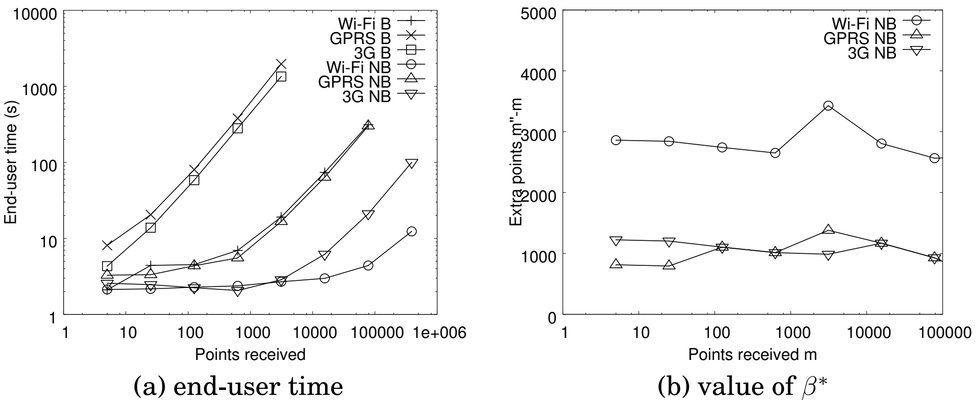


Fig. 24. Effect of the Number of Points Received on the End-User Time and Extra Points Sent

types. Thus, we adopt the Non-blocking method for transferring data objects from the server to the client.

Figure 24b plots the measured value of β^* (i.e., number of unnecessary points sent by the server) as a function of the number of points received by the client. Observe that the value of β^* fluctuates within a certain technology-dependent range $[\beta^-, \beta^+]$. Its range is [800, 1300] for GPRS and 3G, and it is [2800, 3500] for Wi-Fi in our tests. Since the actual value of β^* cannot be predicted, it is not possible to deduce the precise value of m (points received by client) by using the value of m'' (points sent by server). Nevertheless, one may still infer the possible range of m to be: $[m'' - \beta^+, m'' - \beta^-]$. We exploit this information in our privacy analysis in Section 4.2.