Geoinformatica (2013) 17:599–634
DOI 10.1007/s10707-012-0172-9

Blind evaluation of location based queries using space transformation to preserve location privacy

Ali Khoshgozaran · Houtan Shirani-Mehr · Cyrus Shahabi

Received: 28 May 2010 / Revised: 9 June 2012 /
Accepted: 5 November 2012 / Published online: 29 November 2012

Abstract In this paper we propose a fundamental approach to perform the class of Range and Nearest Neighbor (NN) queries, the core class of spatial queries used in location-based services, without revealing any location information about the query in order to preserve users' private location information. The idea behind our approach is to utilize the power of one-way transformations to map the space of all objects and queries to another space and resolve spatial queries *blindly* in the transformed space. Traditional encryption based techniques, solutions based on the theory of private information retrieval, or the recently proposed anonymity and cloaking based approaches cannot provide stringent privacy guarantees without incurring costly computation and/or communication overhead. In contrast, we propose efficient algorithms to evaluate *K*NN and range queries privately in the Hilbert transformed space. We also propose a dual curve query resolution technique which further reduces the costs of performing range and *K*NN queries using a single Hilbert curve. We experimentally evaluate the performance of our proposed range and *K*NN query processing techniques and verify the strong level of privacy achieved with acceptable computation and communication overhead.

Keywords Spatial query processing · Location privacy · Anonymity · Space encoding · Location-based services

A. Khoshgozaran (✉) · H. Shirani-Mehr · C. Shahabi
University of Southern California, 3710 S. McClintock Ave, RTH 323,
Los Angeles, CA 90089, USA
e-mail: jafkhosh@usc.edu

H. Shirani-Mehr
e-mail: hshirani@usc.edu

C. Shahabi
e-mail: shahabi@usc.edu

1 Introduction

The most fundamental classes of location queries predominantly used in location-based services (LBS) are K-nearest-neighbor (*K*NN) and range queries where a group of mobile users want to find the location of their K closest objects from a query point (*K*NN) or all objects located in a certain area of interest (range). An obvious requirement for evaluating *K*NN (range) queries is that the location of the query point (query window) needs to be shared with the location server (server for short) responding to user queries. However, a user's location is highly sensitive information that once compromised, can expose her to various threats such as stalking and inference about her health problems or political/religious affiliations. Therefore, with many LBS applications a user may not want to reveal her location to untrusted and potentially malicious entities (such as the server) in order to preserve her privacy. Such growing concerns about users' privacy in LBS is considered to be the biggest impediment to the explosive growth and popularity of location-based services.

Protecting users' locations while responding to their location-dependant queries is challenging due to an interesting dilemma in resolving such queries: while precise information about query location is needed to generate the result set for a spatial query, the privacy constraints of the problem does not allow revealing users' location information to the potentially untrusted server responding to such queries. In order to resolve this dilemma, we propose a fundamental approach based on utilizing the power of one-way functions and locality-preserving transformations to preserve users' location privacy by encoding the space of all objects and queries and processing spatial queries blindly in the transformed space.

There is an inherent limitation in using traditional encryption techniques for blind evaluation of spatial queries. To illustrate, assume that the server uses recently proposed encryption techniques to compute the encryption of the Euclidean distance between an encrypted point (i.e., the query origin) and each point of interest [14]. These encrypted distances can then be sent back to the client who can decrypt them and find the top K results. Trivially, this protocol satisfies our definition of blind query evaluation (see Section 3) since the location of neither the query point nor the result set is revealed to the server. However, the main limitation here is that distance between query point and each and every point of interest must both be computed and transferred to the client, i.e., $O(n)$ computation and communication complexity where n is the size of the database. There are cryptographic binary search communication protocols that may reduce the communication complexity to logarithmic; however, the computation complexity at the server cannot be further reduced. This is because the points of interest are treated as vectors with no exploitation of the fact that they are points in space.

Another set of approaches to location privacy are K-anonymity and cloaking-based techniques where a user's location becomes indistinguishable among $K-1$ other users or is hidden in a larger *cloaked* region by a *trusted anonymizer* before sending the query to the server. The cloaking and anonymity-based techniques suffer from several drawbacks: (i) By design, these approaches mostly rely on a trusted entity to "anonymize" users' locations which means all user queries are first sent to an *anonymizer* before reaching the server during the system's normal mode of operation. (ii) Users have to extensively trade-off their privacy to achieve higher quality of service or better response time. (iii) The concept of K-anonymity does not protect the user privacy in all scenarios. These issues are discussed in Section 2.

Finally, the solutions based on the theory of private information retrieval achieve strong levels of privacy by ensuring no information about user queries are exposed to the untrusted server. However, private information retrieval is very costly to be employed in practice (see Section 2). Perfect secrecy is achieved at the cost of a linear scan and/or transfer of all database items, or reliance to a hardware device with severely limited computation and storage resources.

In this paper, we utilize space filling curves and one-way hash functions to transform the locations of both user(s) and points of interest into an *encoded* space and to evaluate a query in that space. The transformed space maintains some of the distance properties of the original space which enables efficient evaluation of location queries while remaining irreversible without the knowledge of a *transformation key*. Subsequently, the client can encode the query using its *key* before transferring it to the server and the server blindly responds to the query in the transformed space and returns back to client the encoded answers which are transformed back to the original space using the knowledge of the key at the client side. Consequently, similar to conventional encryption schemes, we do not need any intermediator between the client and server to evaluate the spatial queries *blindly*.

We propose an approximate KNN query algorithm that guarantees constant computation and communication complexity while providing a very close approximation of the original query results and an exact range query algorithm which takes $O(n_l \log T) + |R|$ time where $n_l = \max(n_1, n_2)$ for a rectangular range query of size $n_1 \times n_2$, $T = 2^N$ for N being the Hilbert curve order and $|R|$ denoting the number of objects in the query window. Building on the above two algorithms, we also propose an exact KNN algorithm. Our exact KNN algorithm first computes a candidate result set using the approximate KNN algorithm and uses the results to form a region around the query point guaranteed to include the exact results to the KNN query. It then employs the range query algorithm to retrieve all objects within the computed region.

A preliminary version of this work appeared in [18]. This paper subsumes [18] by

- Utilizing cryptographic hash functions in combination with locality-preserving space filling curves to achieve more stringent privacy guarantees in the presence of sophisticated adversaries with strong prior knowledge about object distributions (Sections 3 and 4).
- Reducing the approximate KNN query complexity from $O(K \times \frac{2^{2N}}{n})$ to $\theta(K)$ for N and n being the Hilbert curve order and total number of objects, respectively (Section 5).
- Proposing blind evaluation of *range queries* as another important class of spatial queries by breaking a 2-D range query into a set of 1-D ranges in the Hilbert transformed space and privately querying the server for the objects in each set (Section 6).
- Employing our approximate KNN algorithm and the proposed range algorithm to enable an exact KNN algorithm (Section 7).

We have experimentally verified the efficiency of our proposed approaches on both synthetic and real datasets. As detailed in Section 10, we show that our approximate KNN algorithm returns a close approximation of the query result in the original space by generating a result set whose elements on average have less than 0.09 mile displacement to the elements of the actual result set in a 26 mile

by 26 mile area containing more than 10,000 points of interest. We also show that with acceptable extra overhead, exact KNN results can be computed using our range query algorithm to retrieve all objects in a region around the query point computed from the results of the approximate KNN algorithm.

The reminder of the paper is organized as follows. We discuss related work in Section 2 and formally define our problem in Section 3. We describe space transformation in Section 4 and subsequently, approximate kNN and exact range query processing in Sections 5 and 6, respectively. Online query processing for exact kNN is presented in Section 7. We introduce our dual curve query processing approach in Section 8 to improve the computation and communication cost of query processing by encoding the space based on two space filling curves instead of one. In Section 9, the end-to-end architecture of query processing is presented. We present our experimental results in Sections 10 and 11 concludes the paper and presents some future directions.

2 Related work

A large body of work in location privacy preserves user location using *cloaking* and K-anonymization techniques [2, 10, 12, 22]. Cloaking and K-anonymity approaches have some limitations. First, by design they rely on a trusted entity to "anonymize" users' locations which means all users posting queries to the location server should trust the *anonymizer* during the system's normal mode of operation. Another limitation of cloaking techniques in general is that either the quality of service or overall system performance degrades significantly as users choose to have more strict privacy preferences. For example, if the user requires a better K-anonymity, the system needs to increase K for that user which would result in a larger cloaked area and hence less accurate query response. Alternatively, if one requires to maintain the quality of service the location server has to resolve the spatial query for each and every point in the cloaked region and send the entire bulky result to the anonymizer to be filtered out. This will obviously affect the overall system performance, communication bandwidth and server throughput and results in more sophisticated query processing. Finally, the concept of K-anonymity does not work in all scenarios. For example, in a less populated area, the size of the extended area can be prohibitively large in order to include $K-1$ other users. Some studies try to address this limitation by proposing more robust ways of determining the area of cloaking [17]. However, they would still need a trusted anonymizer to be able to respond to user queries and in the most optimistic scenario will reveal the region a user is located in, to an untrusted location server. In Section 9, we explain how our proposed approach eliminates the need for an anonymizer.

Recently, *Private Information Retrieval* (PIR) techniques have been proposed to achieve high degrees of privacy while responding to location-dependant queries. Briefly, PIR is defined as follows. A server S holds a database with n bits (or objects), $X = (X_1, \ldots, X_n)$. A user u has a particular index i and wishes to retrieve the value of X_i, without reveling to S the value of i. A PIR algorithm enables u to achieve this goal and secretly retrieve the desired object. The PIR concept was introduced in [5] in an information theoretic setting, requiring that even if S had infinite computational power, it could not find i. PIR can be used to achieve location

privacy by entirely blinding the server from learning any information about what records are being accessed. To employ PIR for privately processing spatial queries, one needs to first organize objects of a dataset into an array of some sort and then replace each traditional retrieval operation needed to process the spatial query with a PIR request. However, PIR is very costly. Using information theoretic PIR, one can achieve perfect secrecy at the cost of a linear client server communication. To reduce this prohibitively expensive cost, computational PIR schemes bring the communication cost down to $O(\sqrt{n})$ by assuming a computationally bounded server where the security of the approaches relies on the intractability of a computationally complex mathematical problem, such as Quadratic Residuosity Assumption [20]. However, similar to information-theoretic PIR, this class of approaches cannot avoid a linear scan of all database items per query. Ultimately, the class of Hardware-based PIR approaches places trust on a tamper-resistent hardware device [1]. These techniques achieve almost optimal computation and communication overhead at the cost of relying on a hardware device (with severe computing and storage resources) to provide perfect secrecy. Therefore, the PIR-based approaches to location privacy [11, 19] are still relatively costly to be employed in practice despite achieving strong user confidentiality.

Space filling curves in general and Hilbert curves in particular have been proposed for range and nearest neighbor query evaluation. For example, Faloutsos and Roseman [9] evaluates the efficiency of Hilbert curves in terms of number of disk accesses for range and nearest neighbor queries. Range query on the images encoded by Hilbert curve is also studied in the image processing community. Chung et al. [6] studied the problem of answering range queries on the Hilbert-scan-based compressed images. However, what distinguishes us from the above studies is our stringent privacy constraints; we want to evaluate spatial queries *blindly* (Section 3). In other words, our main challenge is protecting the very same piece of information (i.e., location information) that is needed by a potentially adversarial entity (e.g., the location server) to effectively respond to location queries.

3 Preliminaries

In this section, we first formally define the problem of blindly evaluating a *K*NN or a range query and briefly discuss our approach and its use of one-way transformations. We also study the challenges associated with finding the right transformations and review an important class of many-to-one dimensional mappings called *space filling curves* which are used in our approach to achieve location privacy.

3.1 Formal problem definition

Given a set of static objects $S = \{o_1, o_2, \ldots, o_n\}$ in 2-D space stored at the location server LS and a set of users $U = \{u_1, u_2, \ldots, u_M\}$ in an area A which can be represented as a set of discrete locations $A = \{l_1, l_2, \ldots, l_{2^{2N}}\}$ (we discretize A into a grid of $2^N \times 2^N$ cells), the *K*NN query with respect to query point q_i finds a set $S' \subset S$ of K objects where for any object $o' \in S'$ and $o \in S - S'$, $D(o', q_i) \leq D(o, q_i)$ where D is the Euclidean distance function. Similarly, a range query returns all objects that fall inside a rectangular query window represented as $w(x, y, n_1, n_2)$

where x and y are the coordinates of the lower left corner of the window and n_1 and n_2 are the height and width of the window query, respectively. As range and KNN queries constitute the dominant spatial queries performed in location-based services, we focus on blind evaluation of these two types of queries and refer to them as *Spatial Queries* hereafter. In a typical spatial query scenario, the static objects represent points of interest (POI) and the query points represent user locations.

Users subscribe to LS to provide them with its location based services. To enable location privacy, a user u_i's location should not be revealed to LS while responding to u_i's queries. Location information can be exposed to LS through a spatial query or its result set. A conventional range or KNN query sent to the server includes location information that can be exploited by the server to pinpoint a user's location. For instance, if u_i is searching for her nearest gas station, her precise location information is needed to find the closest POI. Similarly, LS can deduce u_i's location from the *result* of a spatial query. For instance, suppose u_i manages to secretly receive the gas station o_j as the result of her 1NN query from LS without revealing her location. Here, the server learns that u_i is located somewhere in the *voronoi* cell of o_j. Therefore, our goal is to prevent LS from learning user locations through a query or its result set. We refer to this as the *blind evaluation* criteria.

Definition 1 *Blind evaluation of spatial queries*: Suppose the result of a spatial query q, issued by user u_i located at point l_i evaluates to $R = (o_1, o_2, \ldots, o_K)$ by the server LS. We say q is blindly evaluated if l_i is not revealed to LS via q or R and LS is unable to narrow down u_i's location to a region or among a subset of users.

Note that Definition 1 imposes stronger privacy requirements than the commonly used K-anonymity or cloaking [2, 10, 12, 17, 22, 26] criteria, in which a user is indistinguishable among K other users (where K is usually a very small number) or his location is blurred in a small cloaked region. We note that to the best of our knowledge, achieving perfect secrecy without the usage of PIR is still an open problem. In this paper we avoid the privacy implications of anonymity/cloaking approaches (and their reliance on an anonymizer) and the expensive cost of PIR-based approaches by proposing a fundamental approach of processing queries blindly in a transformed space.

For the rest of the paper, unless otherwise stated, we use the term *user* to refer to a client subscribed to a location-based service located at point l_i issuing the query q. While we always encrypt client/server communications to protect the content of information, anonymous communication is orthogonal to our problem and constructs such as Onion Router (Tor) [7] can be used to protect users against traffic analysis or eavesdropping attacks. Based on the above properties, we term a location server *privacy aware* if it is capable of blindly evaluating spatial queries. The challenge in blind evaluation of spatial queries is protecting the very piece of information (i.e., user location) that is needed to process spacial queries and form a result set. We now show how our notion of *encoding the space* protects users privacy while responding to location-based queries.

3.2 Space encoding

In this section, we introduce our novel approach for protecting user's location from the malicious location servers by transforming the static objects to a new

space using a locality-preserving one-way transformation and also addressing the (transformed) query in this new space. As mentioned earlier, we address the issue of location privacy in the context of location-based services and thus focus on the 2-D space of static objects (i.e., points of interest) and dynamic query points (i.e., user locations). We need a one-way function to map each point from the original space to a point in the transformed space to prevent the server from obtaining the original results by reversing the transformation. A transformation is one-way if it can be easily calculated in one direction (i.e., the forward direction) and is computationally impossible to calculate in the other (i.e., backward) direction. Identifying the right one-way transformation is very challenging because many such mappings do not respect the notion of distance and proximity. The transformations that respect such properties are the only candidates enabling efficient query processing in an encoded space. Therefore, as depicted in Fig. 1, any one-way transformation which respects the proximity of the original space can replace the black boxes in Fig. 1 to make the location server *privacy aware*. Transforming the original space with such a locality-preserving one-way mapping can be viewed as spatially *encrypting* the elements of the 2-D space using a one-way transformation. In practice, many one-way transformations may be reversible even without the knowledge of the trapdoor but the process must be too complex (equivalent to exhaustive try) to make such transformation computationally secure. In this paper we use the properties of our mapping function as the trapdoor only provided to client devices to reverse the encoded results back to their original format while giving the server enough information to process user queries in the encoded space. In a typical LBS setup, the server acts as a service provider and responds to user queries by computing the response in the original 2-D space. However, with our setup, we replace server's data of the original 2-D space with its equivalent *encoded* data in the transformed space and provide the server with means to receive an encoded query information and process it in the transformed space instead. Later in Section 9 we present more details about our overall architecture and show on how careful distribution of this process between client and server helps us achieve private query processing. In the following two sections, we first study the locality-preserving characteristics of space filling curves and proceed to detail our mapping function which uses space filling curves to preserve locality among objects and one-way hashing to make the transformation irreversible.

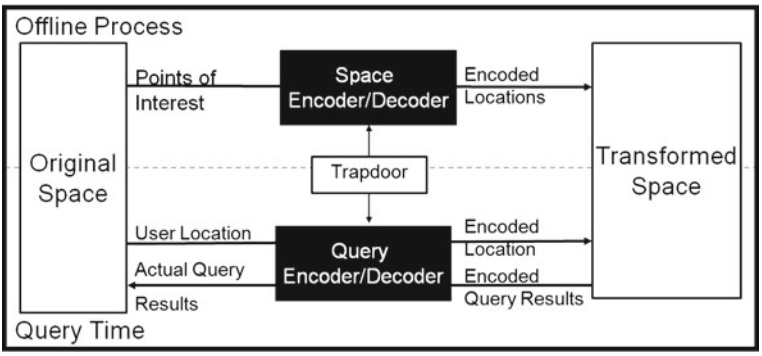


Fig. 1 Space encoding

3.3 Locality preserving space filling curves

We now study an important class of transformations called *space filling curves* as candidate space encoders for our framework and show how they can be treated as space encoders if certain properties of these curves are kept secret from malicious attackers. Introduced in 1890 by an Italian mathematician Peano [25], space filling curves belong to a family of curves which pass through all points in space without crossing themselves. In this way, each point in a multi-dimensional space is assigned to a single point in a one-dimensional space. Some examples of space filling curves are z-curves, Gray-coded curves and Hilbert curves. Examples of these curves are shown in Fig. 2 for a 4×4 grid. The important property of these curves is that they retain the *proximity* and *neighboring* aspects of the data. Consequently, points which lie close to one another in the original space mostly remain close to each other in the transformed space. Examples of space filling One of the most popular members of this class is Hilbert curves [13] since several studies show the superior clustering and distance preserving properties of these curves [9, 15, 21, 23].

Similar to [23], we define H_d^N for $N \geq 1$ and $d \geq 2$, as the Nth order Hilbert curve for a d-dimensional space. H_d^N is therefore, a linear ordering which maps a d-dimensional integer space $[0, 2^N - 1]^d$ into an integer set $[0, 2^{Nd} - 1]$ as follows:

$H = \nu(P)$ for $H \in [0, 2^{Nd} - 1]$, where P is the coordinate of each point in the d-dimensional space. We call the output of this function its *H-value* throughout the paper. Note that it is possible for two or more points to have the same H-value in a given curve.

As mentioned above, our motivating application is location privacy and therefore, we are particularly interested in 2-D space (and hence 2-D curves). Therefore, $H = \nu(X, Y)$ where X and Y are the coordinates of each point in the 2-D space. Figure 3a illustrates a sample scenario showing how a Hilbert curve can be used to transform a 2-D space into H-values. In this example, points of interest (POI) are traversed by a second order Hilbert curve and are *indexed* (transformed to the Hilbert space) based on the order they are visited by the curve (i.e., H in the above formula). Therefore, in our example the points a, b, c, d and e are represented by their H-values 7, 14, 5, 9 and 4, respectively. For any desired resolution, more fine-grained curves can be recursively constructed (Fig. 3b).

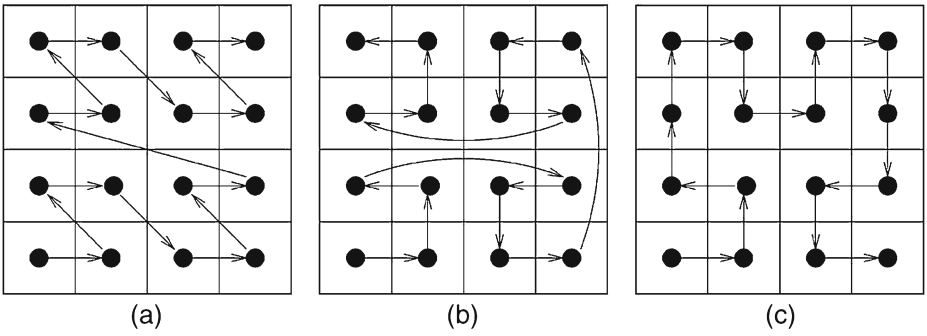


Fig. 2 Space filling curves: z-curve (**a**), Gray-coded (**b**) and Hilbert curve (**c**) [23]

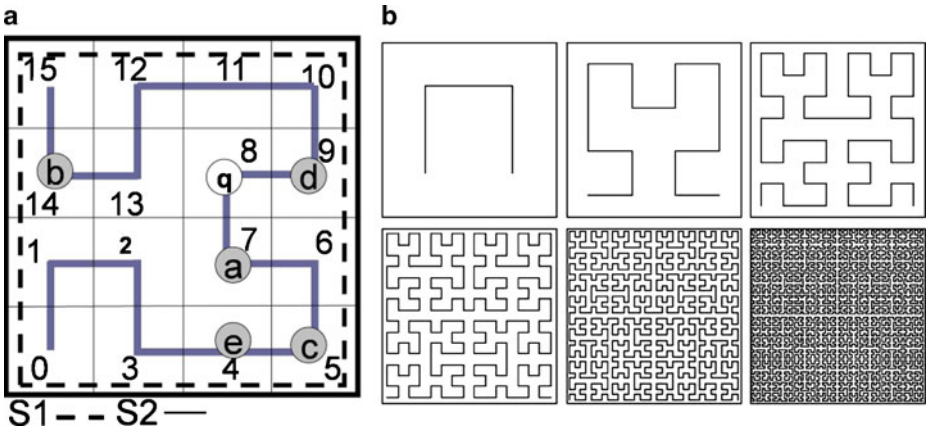


Fig. 3 (**a**) A H_2^2 pass of the 2-D space (**b**) recursive Hilbert curve construction

The key benefit of employing Hilbert curves in our approach is how they can act as locality preserving transformations when used in location-based services. This property suits our approach as our goal is to address spatial queries efficiently in a transformed space. However, as we discussed in Section 3.2 we still need to prevent a hostile entity from reversing such transformations to protect original user locations. We now discuss how we use cryptographic hash functions to achieve this.

3.4 Making Hilbert mapping irreversible

To index objects with a Hilbert curve, one should first choose various curve parameters such as the curve's order (i.e., granularity), starting point, orientation, and scale factor. Let us denote these parameters with N, X_0, Y_0, θ and Γ, respectively. Knowing these parameters, one can effectively compute the H-value of an object with linear complexity (see Section 4.1). However, if these parameters are not known, a substantial brute force effort is required to exhaust all potential combinations to find the right curve parameters. Therefore, protecting the server from learning curve values prevents it from reversing the transformation and obtaining original query parameters using the location information received from the clients. However, this scheme is still susceptible to more sophisticated attacks that employ the untrusted server's prior knowledge of original object distributions. For instance, even without the knowledge of curve parameters, the server can identify a dense area in the transformed space by counting the number of points with similar (or nearby) Hilbert-values. This problem arises from the locality-preserving properties of Hilbert curves and the fact that the untrusted server has access to original Hilbert values and the assignments of points to these values in the transformed space. In this paper, we apply cryptographic hash functions [24] to our Hilbert mappings to make our locality-preserving mapping one-way.[1] With this approach, while the one-way hash

[1] A cryptographic hash allows fast computation of a *digest* in the forward direction while making it infeasible to find the original message given the digest. Moreover, it is infeasible to find two different messages that share the same digest.

function blinds the server from learning any information about object locations, the Hilbert curve ordering allows it to perform spatial queries efficiently in the transformed space and hence satisfying the blind evaluation criteria defined in Section 3.1. The Hilbert curve parameters, together with the hash key, form a *space decoding/encoding key* (SDK) that we use to construct an encoded index for query processing. The challenge, however, is to perform spatial queries efficiently over such an encrypted index without revealing sensitive information to the untrusted server hosting the index. We now elaborate on our approach to achieve privacy while enabling the server to efficiently respond to user queries.

4 Location query processing

Making a query processing engine privacy-aware based on our idea of space transformation discussed above, requires a two-step process consisting of an offline transformation of original space (Section 4.1) followed by online query processing (Sections 5 and 6). The notations used in this section are summarized in Table 1.

4.1 Offline space encoding

During this phase, we first choose the curve parameters (listed in Section 3.3), a nonce ι and a secret key κ. These values together form our transformation key $SDK = \{X_0, Y_0, \theta, N, \Gamma, \kappa, \iota\}$. Next, assuming the entire area covering all points of interest is a square S_1, an H_2^N Hilbert curve is constructed starting from (X_0, Y_0) in a larger square S_2 surrounding S_1 until the entire S_2 is traversed (see Fig. 3a). Let us denote with $|C_{rc}|$ the number of objects in the grid cell C_{rc} located on row r and

Table 1 Notations used in Section 4

Notation	Definition
SDK	Space transformation key set
(X_0, Y_0)	Hilbert curve starting point
θ	Hilbert curve orientation
Γ	Hilbert curve scaling factor
N	Hilbert curve order
C_{rc}	Grid cell located at row r and column c
$(o.x, o.y)$	Coordinates of the object o
$\nu(o.x, o.y)$	Returns the H-value for object o
$o.t$	Object o's textual attributes
ELT	Encoded look up table
ϕ	One-way cryptographic hash function
$\varepsilon_k(x)$	Encrypted value of x
ρ	Average number of objects with the same H-value
prev	Pointer to the previous object based on the H-values
next	Pointer to the next object based on the H-values

Table 2 Sample ELT for Fig. 5

Computed offline	$\phi(9\|1\|\iota)$	$\varepsilon_\kappa(d.t)$	*prev* : a's entry	*next* : b's entry
Stored at the server	5e107d9d…	UZnLpmN…	YQta+/rl…	kjFU4Fq!…

column c. For each object $o_j \in C_{rc}$, $1 \leq j \leq |C_{rc}|$ located at $(o_j.x, o_j.y)$ we construct one entry of an *encoded look up table* ELT defined below.

$$ELT \equiv < \phi(H|j|\iota), \varepsilon_\kappa(o_i.t), prev, next > \ \forall o_i \in S \tag{1}$$

We now discuss in more detail each part of the ELT schema. The term $\varepsilon_\kappa(o_i.t)$ denotes the encrypted value of an object's textual attributes such as name and address represented by $o_i.t$. The | symbol denotes the concatenation operator and $H = \nu(o_j.x, o_j.y)$ returns the H-value for the object o_i using the curve parameters.[2] The ϕ function is a one-way cryptographic hash function such as SHA-1 or MD5 that takes H, an object rank within a cell and a nonce to compute a hash value for each object. This allows the clients to efficiently form an object request during the query processing using SDK. To enable navigation of the encoded index, each entry points to the previous and next object according to the Hilbert ordering (we assign a random order among objects belonging to the same cell). These two pointers are denoted by *prev* and *next* in ELT. For instance, in a cell C_{rc} with three objects ($|C_{rc}| = 3$) and Hilbert value H, the *next* value of the second object points to the third object's entry of the same cell while the *next* value of the third object points to the first object's entry from the next non-empty cell $C_{r'c'}$ with $H' > H$. Finally, to *snap* requests for empty cells to their closest non-empty neighbors, we store one entry per each empty cell and link its *prev* and *next* values to its preceding and succeeding non-empty neighbors in the Hilbert space, respectively.

This process is performed once for the entire object space and at the end of this step, the encoded look-up table ELT is stored at the server. Algorithm 1 (see Fig. 4) details this offline space encoding process. The result of applying Algorithm 1 (see Fig. 4) on object d from Fig. 3a is illustrated in Table 2.

Given a set S of n static objects, let ρ represent the average number of objects with the same H-value in ELT. For a Hilbert curve of order N, $\rho = \frac{n}{2^{2N}}$. Indexing the objects with a lower degree curve is analogous to using a coarse-grained grid. Therefore, using a small N (a large ρ) potentially increases the number of false positives in a query result set. On the other hand, a very fine grained grid (large N) results in many empty cells and hence a storage overhead. Therefore, for a given dataset, we increase N until ρ becomes smaller than a threshold. In Section 10, we experimentally study the effects of the curve order on evaluating range and KNN queries.

[2]We use an efficient bitwise interleaving algorithm from [9] to compute the H-values for points of interest. Depending on the implementation, the cost of performing this operation varies between $O(n)$ and $O(n^2)$ where n is the number of bits required to represent a Hilbert value.

Algorithm 1 CreateIndex(S).

Require: $SDK = \{X_0, Y_0, \theta, N, \Gamma, \kappa, \iota\}$
for all $C_{rc} \in S_2$ **do**
  if $(|C_{rc}| == 0)$ **then**
    $H \leftarrow \nu(r, c)$;
    $ELT \leftarrow ELT \cup < \phi(H|1|\iota), \varepsilon_\kappa(\text{“}dummy\text{”}), prev, next >$;
  end if
  for all $o_j \in C_{rc}, \{1 \leq j \leq |C_{rc}|\}$ **do**
    $H \leftarrow \nu(o_j.x, o_j.y)$;
    $ELT \leftarrow ELT \cup < \phi(H|j|\iota), \varepsilon_\kappa(o_i.t), prev, next >$;
  end for
end for

Algorithm 2 KNN-Generate(K, q_x, q_y)

$H \leftarrow \nu(q_x, q_y)$;
$R \leftarrow$ KNN-Resolve$(\phi(H|1|\iota, K))$;
for all $\varepsilon_\kappa(o_i.t) \in R$ **do**
  $result \leftarrow result \cup \varepsilon_\kappa^{-1}(\varepsilon_\kappa(o_i.t))$;
end for
$return\ result$;

Algorithm 3 KNN-Resolve$(\phi(H|1|\iota, K))$

$idx \leftarrow \phi(H|1|\iota, K)$;
if $isEmpty(idx)$ **then**
  $idx \leftarrow idx.next$;
end if
$qIndexMore \leftarrow idx$;
$qIndexLess \leftarrow idx.prev$;
$R_1, R_2 \leftarrow \emptyset$;
while $|R_1| < K$ **do**
  $R_1 \leftarrow R_1 \cup ELT[qIndexMore]$;
  $qIndexMore \leftarrow qIndexMore.next$;
end while
while $|R_2| < K$ **do**
  $R_2 \leftarrow R_2 \cup ELT[qIndexLess]$;
  $qIndexLess \leftarrow qIndexLess.prev$;
end while
$R \leftarrow R_1 \cup R_2$;
$return\ R$;

Fig. 4 Algorithms CreateIndex (*top*), KNN-generate (*middle*) and KNN-resolve *(bottom)*

5 Online query processing-approximate *K*NN

Using the encoded index ELT, we show how *K*NN queries are blindly evaluated in the transformed space. Algorithm 2 (see Fig. 4) details the steps taken while responding to a user's *K*NN query. The basic idea is to first encode the static objects once (CreateIndex) and then to calculate, during the query time, the accurate starting point on the Hilbert curve (KNN-Generate) and to expand the search in either direction until the candidate result set is found (KNN-Resolve). To elaborate, for each query point q located at position (q_x, q_y), KNN-Generate uses SDK to

compute $H = \nu(q_x, q_y)$. We then use KNN-Resolve (Algorithm 3 (see Fig. 4)) to expand our search in the encoded space starting from the first entry $\phi(H|1|\iota, K)$. Note that since Hilbert values are hashed before being stored at the server, we cannot compute the distance between each point in the encoded space. Therefore, the search has to expand to K objects in each direction. While storing original Hilbert values would allow comparisons among object distances (at least in the Hilbert space), as we discussed in Section 3.4, it would make our scheme vulnerable to server's prior knowledge attacks. The computation and communication burden of searching for K more objects is the cost we pay to achieve more stringent privacy. Next, knowing SDK, KNN-Generate uses ε_κ^{-1} to decrypt the response received from KNN-Resolve result set back to the original space. To illustrate, in our example, having $K = 2$, and $q = (2, 2)$, KNN-Generate computes $H = 8 = \nu(2, 2)$ and calls KNN-Resolve $(\phi(8|1|\iota, 2))$ to obtain $R = \{\varepsilon_\kappa(d), \varepsilon_\kappa(b), \varepsilon_\kappa(a), \varepsilon_\kappa(c)\}$. Next, ε_κ^{-1} is applied to objects in R to obtain their original attributes. In Section 9, we show how appropriate placement of these modules on client and server sides enables blind evaluation of spatial queries.

We can now derive the complexity of the KNN-Resolve module which represents the overall KNN query processing complexity. Using ELT instead of the curve itself, we only visit Hilbert values that contain at least one object. Therefore, Algorithm KNN-Resolve visits at most $\theta(2 \times K)$ entries in ELT. Furthermore, our communication complexity is determined by average number of objects read and transferred to the user which is $2\rho K$ also being constant since $\rho \approx 1$ (see Section 10.1).

It is important to note that the generated query result set is approximate due to the dimension reduction technique used. For instance, in Fig. 3a, while e is closer to q than b, it is not included in the result of the 2NN query. Later in Section 8, we propose an efficient technique that significantly reduces the approximation error for the method discussed above. We also present an exact KNN algorithm in Section 7 which is relatively more costly than our approximate results but returns the actual closest objects. We conduct several experiments in Section 10 to evaluate the accuracy of our approximate KNN and the cost of our exact KNN methods. While the approximate results are fully practical in an LBS scenario, exact results can be derived with reasonable extra overhead.

6 Online query processing-range

During the offline process described in Section 4.1, the objects in the original space are encoded and their H-values are stored in ELT. Our general approach to answer a 2-D range query $w(x, y, n_1, n_2)$ is to transform it to a series of 1-D ranges in the Hilbert transformed space and the challenge is how to do this transformation efficiently and accurately. Figure 5b shows an example of a range query $w(1, 2, 3, 6)$. The result of this query contains the highlighted objects o_1 and o_2 in Fig. 5b whose H-values are contained in $w(1, 2, 3, 6)$. As the figure illustrates, $w(1, 2, 3, 6)$ can be transformed into five 1-D range queries: objects whose H-values range from 8 to 13, equals 17, range from 30 to 33, equals 46 and finally, from 50 to 55. Each of these 1-D range queries is called a range query run or a *run* for short (a similar notion of runs is also defined by Moon et al. in [23]). These runs are highlighted in Fig. 5b. The

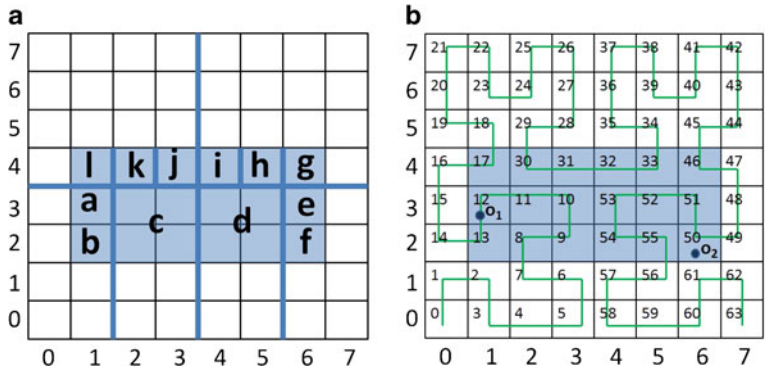


Fig. 5 A range query decomposed into 12 maximal blocks (**a**), with the underlying Hilbert curve (**b**)

result of the range query consists of objects whose H-values belong to any of these two runs (o_1 and o_2, with H-values 12 and 50 in our running example).

6.1 Finding the range query runs

In order to find the query runs, we adopt the method proposed in [6] and show how this method can efficiently respond to a range query. To find the range query runs, the first step is to decompose the input range query into a set of square blocks according to the quadtree decomposition. During this process, the grid space is recursively divided into four equal partitions until a partition is completely contained in the range query. For example in Fig. 5a, the square labeled c is obtained after two recursions while acquiring the squares a and b needs a third recursion.

After decomposing the range query into a set of square blocks according to the quadtree decomposition, the resulted square blocks are named *maximal quadtree blocks* or *maximal blocks* for short [6]. An interesting property of maximal blocks is that the H-values inside a maximal block form a continuously increasing sequence. The minimum and the maximum H-values of the sequence are denoted as h_b and h_e, respectively (the letters b and e are used to denote the *beginning* and *end* of each sequence). Later, we show that we can efficiently find the values of h_b and h_e and consequently, find all the H-values inside a maximal block. Similar to [6], we denote a maximal block as $MB(x, y, s)$ where (x, y) and s are the lower left coordinate and the side length of the maximal block, respectively. For instance, as illustrated in Fig. 5a, $MB(2, 2, 2)$ is a maximal block of the range query $w(1, 2, 3, 6)$ whose H-values form a continuously increasing sequence with $h_b = 8$ and $h_e = 11$ (h_b and h_e are the minimum and maximum H-values within $MB(2, 2, 2)$). We denote each sequence as a pair in the form of (h_b, h_e). Hence, the H-values inside $MB(4, 2, 2)$ can be denoted as $(52, 55)$.

To find the maximal blocks of a range query, we use the *strip splitting based* optimal algorithm proposed by Tsai et al. in [27]. The algorithm can find the maximal blocks of a $w(x, y, n_1, n_2)$ range query in $O(n_l)$ time where $n_l = \max(n_1, n_2)$. The idea is to repeatedly split a strip of maximal blocks from the sides of the range query. After decomposing the range query into a set of maximal blocks, we need to find the values

of h_b and h_e for each maximal block. Calculation of these values is explained in detail in [6] and are skipped here due to the lack of space.

The H-values obtained from the maximal blocks of a range query form a sequence called *Seq*. For example, the range query in Fig. 5b results in the following sequence:

$$Seq = \{(13, 13), (12, 12), (50, 50), (51, 51), (8, 11), (52, 55), (17, 17),$$
$$(30, 30), (31, 31), (32, 32), (33, 33), (46, 46)\}.$$

The H-values in *Seq* generated by the strip-based decomposition algorithm may not be in increasing (or decreasing) order. Therefore, we sort the H-values in *Seq* and denote it as Seq^*. With Seq^*, we can *merge* adjacent pairs which belong to the same run in order to decrease the total number of runs and instead, increase their length. We can merge a pair p in Seq^* with its successor q, if and only if the difference between h_b of q and h_e of p is one. The sequence obtained by merging the elements of Seq^* constitute the runs of the range query and we denote the sequence as $runs(w)$. For the above example, $runs(w)$ is as follows:

$$runs(w) = \{r(8, 13), r(50, 55), r(30, 33), r(17, 17), r(46, 46)\}.$$

Each element of $runs(w)$ is a run in the original range query consisting of H-values from x to y denoted as $r(x, y)$. For example, in Fig. 5b, the query has five runs: $runs(w) = \{r(8, 13), r(50, 55), r(30, 33), r(17, 17), r(46, 46))\}$. Sorting and merging the H-values greatly reduce the number of range query runs. For example, we achieved a 74 % reduction in number of runs for randomly generated 200,000 range queries with various side lengths.

We have so far showed how a range query is converted to a set of runs that consist a continuous increasing sequence of Hilbert values. To privately query the objects in each run, we need to convert each run to its private representation. In order to do that, each run $r(H_s, H_e)$ is first converted to $r(\phi(H_s|1|\iota), \phi(H_e.next|1|\iota))$. With this representation, we guarantee that all objects whose H-values fall within r are returned as part of the range query.

While evaluating a range query, we need to include all cells that overlap with the query window. As some of these cells only partially overlap with the query, this process might introduce some *excessive objects* in the query result set. However, we show in Section 10, that excessive objects constitute a marginal fraction of the objects retrieved from ELT and therefore can be easily filtered by the users without affecting the client/server communication cost.

We now discuss the time complexity of the quadtree decomposition algorithm. As mentioned above, the process to find the range query runs $w(x, y, n_1, n_2)$ consists of four parts: (1) decomposing w into its maximal blocks, (2) finding the value of h_b and h_e for each maximal block and forming the set *Seq*, (3) sorting the elements of *Seq* and forming Seq^*, and (4) merging the elements of Seq^* and forming $runs(w)$. The first step can be done in $O(n_l)$ as discussed earlier. The second step can be done in $O(n_l \log T)$ where $T = 2^N$ for N being the curve order [6]. The third step can be done in $O(n_l \log n_l)$ using a sorting algorithm such as quick sort. Finally, the forth step can be done in $O(n_l)$. Consequently, the total running time to find the range query runs $w(x, y, n_1, n_2)$ is $O(n_l \log T)$. The last step of the range query processing is to retrieve relevant objects from ELT. There are on average $n_1 \times n_2 \times \rho$ many objects in a range query of size $n_1 \times n_2$ which results in equal number of reads from ELT.

Algorithm 4 Range-Generate(x, y, n_1, n_2)

Require: $SDK = \{X_0, Y_0, \theta, N, \Gamma, \kappa, \iota\}$
1: $w_{runs} \leftarrow FindRuns(x, y, n_1, n_2)$;
2: **while** $w_{runs} \neq \emptyset$ **do**
3: $< run_s, run_e > \leftarrow w_{runs}.GetNextRun()$;
4: $H_s \leftarrow \nu(run_s)$; $H_e \leftarrow \nu(run_e)$;
5: $R \leftarrow$ RangeResolve$(\phi(H_s|1|\iota), \phi(H_e.next|1|\iota))$;
6: **while** $R \neq \emptyset$ **do**
7: $o_i.t \leftarrow \varepsilon_\kappa^{-1}(r.GetNextObj())$;
8: $result \leftarrow result \cup o_i.t$;
9: **end while**
10: **end while**
11: $return\ result$;

Algorithm 5 Range-Resolve(s, e)

1: $idx \leftarrow s$;
2: **if** $isEmpty(idx)$ **then**
3: $idx \leftarrow idx.next$;
4: **end if**
5: **while** $idx \neq e$ **do**
6: $R \leftarrow R \cup ELT[idx]$;
7: $idx \leftarrow idx.next$;
8: **end while**
9: $return\ R$;

Fig. 6 Algorithms range-generate (*top*) and range-resolve (*bottom*)

6.2 Online range query processing modules

We already discussed the details of blind range query evaluation in Section 6.1. Here, we describe Range-Generate and Range-Resolve, the two modules involved in online range query processing (Algorithms 4 and 5 (see Fig. 6)). For each range query $w(x, y, n_1, n_2)$, Range-Generate first finds the runs of w by calling FindRuns(x, y, n_1, n_2). FindRuns uses the techniques discussed in Section 6.1 to return a set of runs corresponding to the range query. For each run, Range-Generate privately retrieves the objects corresponding to the run using the Range-Resolve module. The Range-Resolve module privately returns all objects that belong to a run. Range-Generate then decrypts the objects retrieved in the response using SDK and generates the actual query result set. Observe that mapping a range query to its set of runs only requires the knowledge of SDK and does not depend on the distribution of static objects. Therefore, the runs representing each range query can be calculated without requiring any knowledge of the distribution of static objects being queried.

7 Online query processing-exact *K*NN

As we already discussed in Section 5, our proposed private *K*NN query processing algorithm is approximate. To illustrate how we can enable exact *K*NN queries using

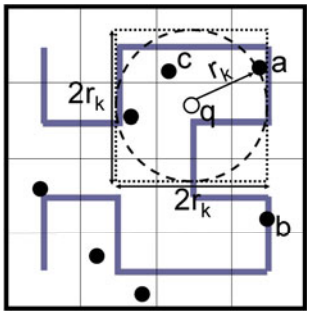


Fig. 7 Exact *K*NN query processing

approximate *K*NN and range query processing modules defined in the previous sections, consider the example in Fig. 7. Given a *K*NN query centered at q, we first compute the approximate answer using Algorithms 2 and 3 (see Fig. 4). In our running example, for $K = 1$ this step results in $result = \{a, b\}$ or $1NN(q) = a$. However, this result is approximate since the actual nearest neighbor of q is c. Let us denote by r_k the distance between q and its Kth closest object according to the approximate *K*NN Algorithm. Let us use C_k to represent the circle centered at q with radius r_k. To obtain the exact *K*NN result, all objects in C_k should be retrieved as their distance to q is less than r_k. Therefore, we form a $2r_k \times 2r_k$ square centered at q that is guaranteed to include all objects in C_k and use Algorithms 4 and 5 (see Fig. 6) to retrieve its enclosing objects. In Fig. 7, this process results in retrieving 2 new objects including c as the actual nearest neighbor of q.

In summary, to achieve exact *K*NN results, an approximate *K*NN query is followed by a range query that is guaranteed to return all objects that might be potentially closer to q than the result set of the approximate *K*NN algorithm. This process can however, be costly for two reasons. First, the approximate nature of the *K*NN algorithm might result in a relatively large C_k. Secondly, such large C_k can result in multiple query runs that need to be further processed. In the following Section, we introduce our proposed *dual curve query processing* technique that reduces both of these costs. By improving the *K*NN query accuracy, we first reduce r_k which results in a smaller region to be passed to our range query module. In addition, we use our proposed technique to reduce the number of query runs in the range query window. Therefore, our proposed technique (i) improves the accuracy of approximate *K*NN results (ii) reduces the communication cost in a range query and therefore, (iii) improves the computation and communication cost of our exact *K*NN algorithm. We proceed to present the details of our approach and how to improve the performance of our spatial query processing techniques discussed so far.

8 Dual curve query resolution

Using a single Hilbert curve as a space encoder for processing approximate *K*NN and range queries discussed in Sections 5 and 6 has several drawbacks for each algorithm. In this section, we first discuss these drawbacks and then introduce our *Dual Curve Query Resolution* approach or *DCQR* which overcomes the weaknesses of the former scheme and generates significantly more satisfactory results for both

range and approximate *K*NN queries. As we noted before, such improvements will directly affect the exact *K*NN algorithm as well. We study how *DCQR* improves approximate *K*NN, exact range, and hence, exact *K*NN algorithms in Sections 8.1, 8.2 and 8.4, respectively.

8.1 Approximate *K*NN queries: proximity in Hilbert curves

A closer study of Hilbert curves reveals two important properties of such curves that can significantly affect the performance of the approximate *K*NN algorithm proposed in Section 5 (for ease of read, throughout the remainder of Section 8 we simply omit the word "approximate" when referring to approximate *K*NN query processing and instead explicitly use the term "exact" to refer to non-approximate *K*NN algorithm). First, consider the 1st degree curve of Fig. 8a. The curve is constructed by traversing a U-shaped pattern. Regardless of its orientation, such a curve fills the space at a specific direction at any given time sweeping the space in a clockwise fashion. Starting from the first degree curve of Fig. 8a, the curve misses one side in its first traversal. As the curve order grows, the number of missed sides grows exponentially as well so that an H_2^N curve misses $M = 2^{2N} - 2^{N+1} + 1$ sides of a $(2^N - 1) \times (2^N - 1)$ grid. The above property makes H-values of certain points farther as N increases. For instance the Euclidean distance between points a and d is similar to the that of points b and c in the original 2-D space, however, as N grows, due to the above property the difference between a and d's H-values grows exponentially larger than that of b and c. Therefore, points closer to two quadrants of the space (i.e., the first and last quadrants filled by the curve) will be *spatially furthest* from one another in the transformed space.

The second drawback of using a single Hilbert curve for *K*NN queries is due to the fact that such space-filling curves essentially reduce the dimensionality of the space from 2 (or in general case N) to 1. Naturally, each element in the 1-D space constructed by the Hilbert curve will have two nearest neighbors compared to the original case where each element (except those at the edges) has four (or in general case $2N$) nearest neighbors. Therefore, as [16] suggests, in the best case scenario, only half of these nearest neighbors in 2-D space will remain a nearest neighbor of the same point in the transformed 1-D space. It is clear to see how each of these issues play a negative role on the quality of the result set given *K*NN's sensitivity to how the underlying index structure preserves object proximity.

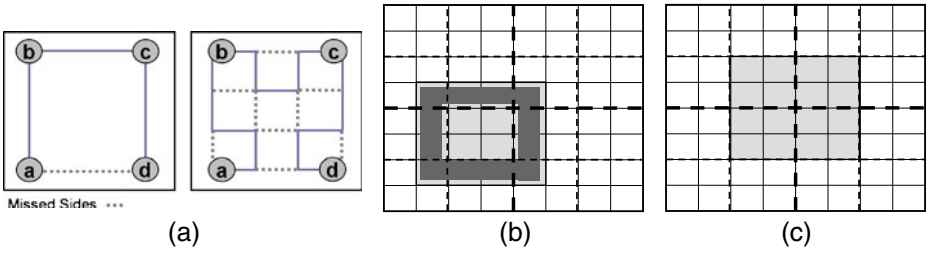


Fig. 8 Missed sides of H_2^1 and H_2^2 curves (**a**), four strips around the range query (**b**), shifted range query (**c**)

8.2 Range queries: the number of query runs

In Section 6, we showed how a range query is blindly evaluated by generating the range query runs and looking up ELT for all encoded objects whose H-values are located inside each run. Since ELT is stored at the server side, all requests for runs have to be transferred to the server. Therefore, it is desirable to reduce the number of runs in a range query to minimize the communication cost of a range query request and the server's throughput in responding to client requests.

Consider the following example. Given a range query $w(x, y, n_1, n_2)$, assume x and y to be odd and n_1 and n_2 to be even numbers. The range query has four strips each consisting of only maximal blocks of size 1×1 i.e., four strips intersecting in the four corners of the range query. An example of this situation is shown in Fig. 8b for $w(1, 1, 4, 4)$, where strips consist of 12 1×1 maximal blocks. It is easy to verify that if at least one of x and y becomes even, while n_1 and n_2 remain the same, the number of maximal blocks decreases substantially as some of the strips disappear. For example in Fig. 8c both x and y are even numbers and there are no strips in the range query. A formal proof of the relationship between the number of quadtree blocks and the query window location can be found in [8]. As mentioned in Section 6, range query runs are obtained by merging the H-values resulted from maximal blocks of the range query. This gives us the intuition that larger number of maximal blocks results in more query runs. We experimentally investigated this and the result confirms our intuition. Therefore, any technique that reduces the number of maximal blocks will consequently result in less number of query runs.

Finally, it is easy to observe how the combination of the dimension reduction properties of Hilbert curves in evaluating approximate KNN queries and the excessive number of runs generated from a range query can negatively affect the performance of our exact KNN algorithm as well. We now present a solution to rectify the aforementioned shortcomings.

8.3 A dual curve for query processing

Using a single curve to encode the objects in space results in a loss of precision (and thus a negative effect on overall quality of returned results) for KNN queries and server overhead for range queries. We mitigated both of these issues by creating a *dual* curve which is a replication of the original curve rotated 90 degrees and shifted by one unit in both x and y directions and index the objects using both curves. As we show in this section, applying the rotation operator substantially improves KNN query results compared to the shift operation. Conversely, applying the shift operator to an original curve reduces the server throughput by decreasing the number of query runs while the rotation operation does not significantly affect the efficiency of our range query processing algorithm.

The rotation operator has a positive effect on KNN query evaluation since by rotating the degree N curve, all lower degree curves constructing the main curve will also be rotated. At each curve order, the curve rotation ensures that the missed sides generated by the discontinuation of the curve (such as the missed sides between points a and d in Fig. 8a), will be covered by the rotated curve. Therefore, the points deemed spatially far from each other in one curve will be indexed correctly in the other curve. This will address the first issue of using a single Hilbert curve for

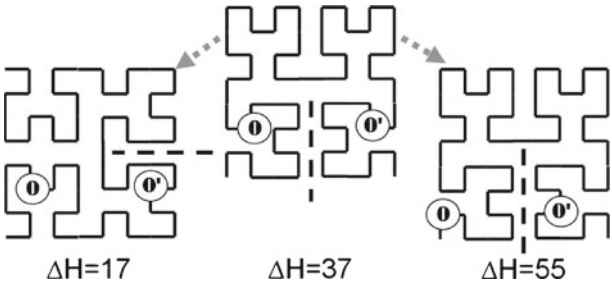


Fig. 9 Proximity in the original curve (*middle*) vs. the rotated (*left*) and shifted (*right*) dual curves

*K*NN queries. A rotated dual curve also mitigates the effect of the second property discussed above by transforming the 2-D space to two 1-D spaces. Therefore, each point will now have two nearest neighbors in each curve note that these two neighbor pairs can (and often do) have overlaps. However, adding the second curve results in more accuracy for *K*NN query results. Although a shifted-only dual curve can also cover many gaps in the original curve, it cannot solve the first shortcoming discussed for *K*NN queries because a one unit shift cannot bring objects with large H-value differences closer to each other in the dual curve as a shift does not change the curve orientation (See Fig. 9).

Similar to the above discussion, it is easy to show the positive effect of a shift operator for range query processing. Since applying the rotation operator on the dual curve does not change the number of quadtree blocks (and thus the number of query runs), we only need to study the effect of the shift operator on the number of query runs. Shifting the original curve with the translation vector (1, 1), can cause the strips with maximal blocks of size 1×1 to disappear. Note that, in some situations, a shifted curve might remove some strips around the range query while introducing new ones. One can verify that the above scheme will reduce the total number of maximal blocks in almost $\frac{5}{16}$ of the cases for square range queries (roughly similar improvements can be achieved for arbitrary rectangular range queries). This ratio can be gained by considering the 16 possible configuration of even or odd numbers representing a window query's x, y, n_1 and n_2 values. Therefore, using a shifted and rotated dual curve, we can reduce the number of maximal blocks of a range query which causes a decrease in the number of query runs.

8.4 Exact *K*NN queries: smaller regions with less runs

It is now easy to observe how exact *K*NN query processing benefits from $DCQR$. The advantages are twofold. For one, $DCQR$ substantially improves the probability of finding a closer *K*th nearest object (see Section 10.3) which in turns results in a smaller region that needs to be fetched using the range query. Moreover, given a region formed by the approximate *K*NN algorithm, $DCQR$ frequently reduces the number of runs resulted from the region by identifying the curve that results in less number of runs. Therefore, the combination of $DCQR$ effects on approximate *K*NN and range algorithms significantly improves the overall performance of exact *K*NN while avoiding a significant performance penalty. We study the performance of all three classes of algorithms in Section 10.

Using the $DCQR$ approach, we need to slightly modify the offline index construction 4.1 and accordingly adjust the spatial query processing logic for range and nearest neighbor algorithms. During the offline phase, we now need a new key SDK' for the dual curve which is computed as $SDK' = \{X_0 + 1, Y_0 + 1, \theta + 90°, N, \Gamma, \kappa', \iota\}$ from SDK. Consequently two Hilbert curves H_2^N and H'^N_2 are constructed based on SDK, SDK' and visiting each point, we construct two records for ELT and ELT' as detailed in Section 4.1.

The query processing also follows the logic from Sections 5, 6 and 7. While evaluating KNN queries, for each query point q, we compute $H = \nu(q_x, q_y)$ and $H' = \nu'(q_x, q_y)$ using SDK and SDK', respectively. We then initiate two parallel query resolution schemes using both ELT and ELT' simultaneously to retrieve K closest matches for each curve separately. Similar to Section 5, we decode the two result sets and choose the K best candidates (based on their Euclidean distance to q). As for a range query, we generate two parallel queries one for each curve and pick the one that results in fewer runs. Furthermore, with regard to complexity, the $DCQR$ query processing computation and communication cost for KNN queries is almost twice as much as Algorithm 3 (see Fig. 4) due to traversing two curves and returning twice as many points in the result set. For range queries, using the $DCQR$ approach slightly increases the computation cost (due to considerable overlap among computing runs for two different curves) while reducing the communication cost (due to selecting the curve resulting in fewer runs). We experimentally measure these costs and benefits in Section 10.

9 Proposed end-to-end architecture

We are now ready to explain how our space encoding technique and careful placement of KNN and range query evaluation modules in client and server sides enable blind evaluation of spatial queries in location-based services. The *client* (e.g., a user's portable device) issues a spatial query and provides the necessary parameters for each query type. In order to make the location server privacy-aware, we assume the architecture of Fig. 10, which details the offline indexing and online query processing

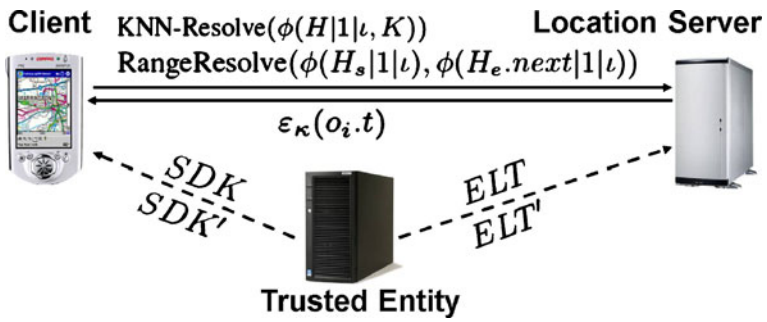


Fig. 10 *DCQR* architecture for spatial query processing

of a spatial query and use the algorithms discussed in Section 8 to modify the classic location-based services architecture in the following three ways:

(1) A *trusted entity* is added to the architecture to perform the CreateIndex module once and to create and populate ELT and ELT' with POI's encoded information. A second functionality of the trusted entity is to securely embed (SDK, SDK') key pair required to encrypt (decrypt) the query (results) into client devices. Note that the key pair is the same among all users to prevent redundant calculations at the query processing time. However, to avoid collusion attacks among malicious users and the server, the keys are stored in inexpensive smartcards placed in subscriber's client devices [3] allowing encoding/decoding of data securely without sharing the key with actual end-users. Therefore, users generate (receive) queries (results) in plaintext while not having access to any intermediate calculated Hilbert values. In other words, we place trust on a tamper resistent commodity hardware as opposed to potentially malicious users. Moreover, by applying techniques such as timestamping [4] we prevent replay attack during query processing. With timestamping, the freshness of the message is verified by checking the timestamp of each message. Finally, the trusted entity provides the location server with the two encoded indexes ELT and ELT' instead of the original data set and keeps the key pair secret from the location server. Note that as opposed to anonymization and cloaking techniques discussed in Section 2, the trusted entity performs the above actions only once $offline$ and thus is not involved in the online query processing schemes.
(2) To enable blind evaluation of range and KNN queries, we let clients execute the KNN-Generate and Range-Generate modules that require the knowledge of SDK, SDK' to form encrypted requests for the server. The server is in charge of executing the KNN-Resolve and Range-Resolve modules both of which only require access to the encrypted indexes ELT and ELT'. Notice that as described above, the smartcard on the client device securely transforms a range (kNN) query into a set of query runs (Hilbert values) which are sent to the server to enable query processing. This prevents a malicious user from obtaining the transformation keys during query processing.
(3) To execute exact KNN queries, clients first issue an approximate KNN query and use its results to form a subsequent range query that retrieves a set of objects guaranteed to contain the exact results to the original query (Section 8.4). Once the results are returned to the client, it uses SDK, SDK' to decrypt the result to obtain original object information.

10 Experimental evaluation

We have conducted extensive sets of experiments to evaluate the performance of our proposed approaches. The effectiveness of $DCQR$ is determined in terms of (i) the effect of the curve order N on range and KNN queries (ii) accuracy and efficiency of our proposed methods for approximate KNN and range queries and (iii) the performance of exact KNN query processing. Note that as we detailed in Sections 1 and 2, our one-way locality preserving transformation allows us to obtain more stringent privacy guarantees compared to other approaches discussed in Section 2. However, we were unable to empirically compare our techniques with these studies

(a) Uniform (b) Real-world (c) Skewed

Fig. 11 Datasets

because they mostly evaluate performance based on the size of the K-anonymity set or the cloaked region and the effectiveness of the anonymization techniques used while these metrics are meaningless in our approach.

Our experiments are performed on three different datasets: (i) a synthetically generated uniform dataset (ii) a real-world dataset of restaurants obtained from NAVTEQ (www.navteq.com) covering a 26 mile by 26 mile area surrounding the city of Los Angeles and (iii) a synthetically generated skewed dataset where 99 % of the objects form four Gaussian clusters (with $\sigma = 0.05$ and randomly chosen centers) and the other 1 % of the objects uniformly distributed (Fig. 11). All three datasets contain around 10,000 objects (n =10,000). Experiments were carried out on an Intel $P4$ 3.20 GHz with 2 GB of RAM and are implemented in Java.

10.1 Choosing the right curve order (N)

As our first set of experiments, we evaluate the effectiveness of our proposed indexing technique by analyzing the curve behavior for different values of N (i.e., curve order) and choosing the right value for the rest of our experiments. Using two H_2^N curves for indexing objects, we measure the average number of objects which are assigned the same H-value (i.e., ρ) for each value of N for our three datasets. Ideally we want to have $\rho \approx 1$. Figure 12 shows how ρ changes with N and quickly approaches 1 for $N \geq 10$ for all of our three datasets. Due to the extra overhead of an unnecessary increase in N for range and KNN queries, we choose the smallest value of N that results in $\rho \approx 1$. Therefore, for the rest of our experiments, we set $N = 12$ unless otherwise stated.

10.2 Evaluating query results' quality

Suppose the ground truth result of a query q, issued by a user located at point l_i is $R = (o_1, o_2, \ldots, o_K)$ if no privacy is required. Let us denote by $R' = (o'_1, o'_2, \ldots, o'_{K'})$ the query result obtained using our proposed techniques. Given the nature of our approximate KNN algorithm and the fact that our range algorithm might append extra objects to the result set, we use the following two metrics to evaluate the quality of results.

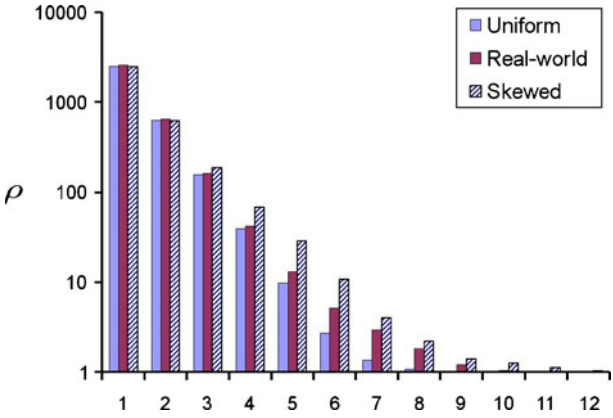


Fig. 12 Finding the right value of curve order (N)

Precision By precision, we measure what fraction of the points retrieved are relevant for range and approximate *K*NN queries. This metric is defined in Eq. 2 ($|R|$ denotes the cardinality of the set R).

$$Precision = \frac{|R \cap R'|}{|R'|} \tag{2}$$

For our approximate *K*NN algorithm, we introduce an additional *displacement* metric defined below (note that we do not approximate range queries).

Displacement Measuring how *closely* R is approximated by R' using our *K*NN algorithm:

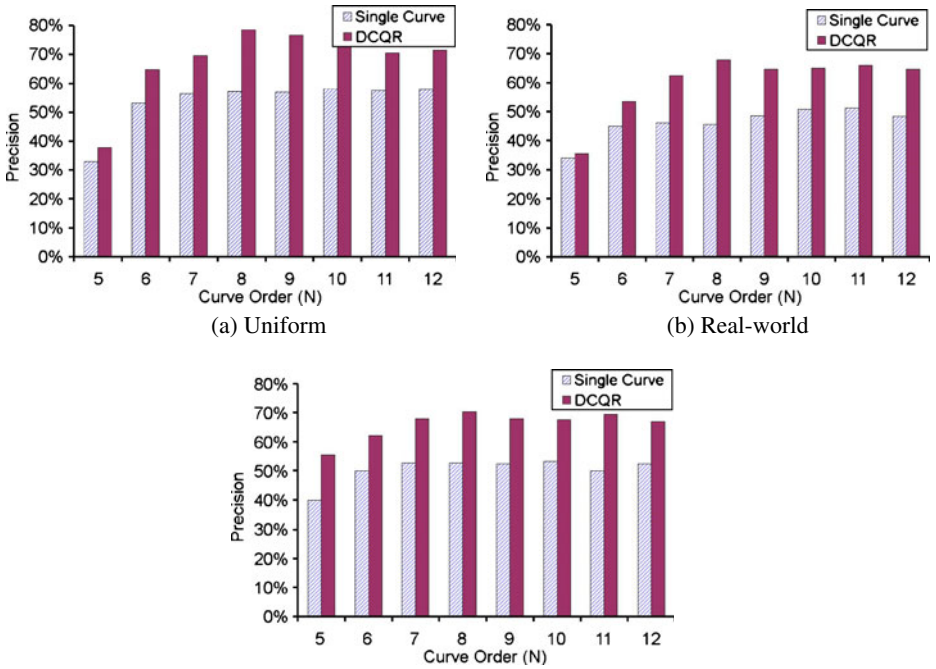
$$Displacement = \frac{1}{K}\left(\sum_{j=1}^{K} ||l_i - o'_j|| - \sum_{j=1}^{K} ||l_i - o_j||\right) \tag{3}$$

where $||l_i - o_j||$ is the Euclidean distance between the fixed query point l_i and an object o_j. Obviously, since R is the ground truth, *displacement* ≥ 0. In the remainder of this section, we use the above metrics to experimentally evaluate our range and *K*NN algorithms.

10.3 Approximate KNN query evaluation

The next set of experiments evaluates the performance of our approximate *K*NN query processing algorithm for different values of N and K for our three datasets by comparing the single curve approach against DCQR. In the remainder of this section, each experiment is performed for 1,000 randomly generated queries and the results are averaged.

As illustrated in Fig. 13, DCQR results in an average 15 % improvement in precision over the single curve approach (from 50 to 65 %). It is also clear from the figure that for all three datasets the precision quickly reaches 70 % for $N = 8$ (i.e., $\rho < 2$) and slightly oscillates for larger values of N. Finally, as expected, the uniform dataset achieves slightly higher precision compared to our real-world and



(a) Uniform (b) Real-world

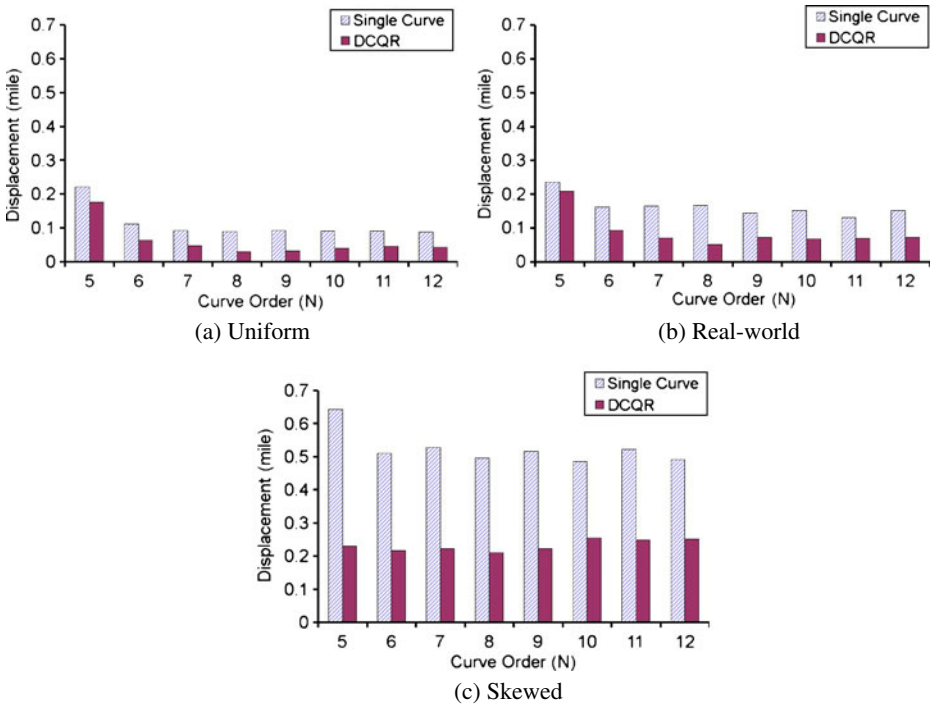
(c) Skewed

Fig. 13 Precision vs. curve order (N)

skewed datasets. Figure 14 also illustrates how DCQR reduces the displacement error by 50 % for all three datasets. Also, as expected, the DCQR displacement is slightly higher in the skewed dataset. Note that the displacement is a more reasonable measure for *K*NN query evaluation compared to the precision because it shows how *close* the query results are approximated rather than just showing the percentage of exact neighbors in the result set. For instance even a 0 % precision with 0.05 mile displacement means that although no exact match for the query is found, each approximate result is less than 0.05 mile farther from the query point than the actual result.

Finally, we measured the $DCQR$ overhead in overall approximate *K*NN query response time. As Fig. 15 illustrates, the overhead stays around 2 milliseconds on average for the three datasets. Also the response time stays around 6 milliseconds for all three datasets.

The next sets of experiments are identical to what we discussed above except that we now study how approximate *K*NN query processing is affected by varying K and fixing curve order at $N = 12$. As Fig. 16 illustrates, DCQR improves precision by more than 25 % for all three datasets averaging 67 %. The displacement also increases linearly with K while DCQR results in more than 50 % error reduction (Fig. 17). Similar to the former set of experiments, the skewed dataset results in relatively larger displacements compared to the other datasets. Finally, Fig. 18 shows how response time linearly grows with K, confirming our query complexity



(a) Uniform (b) Real-world

(c) Skewed

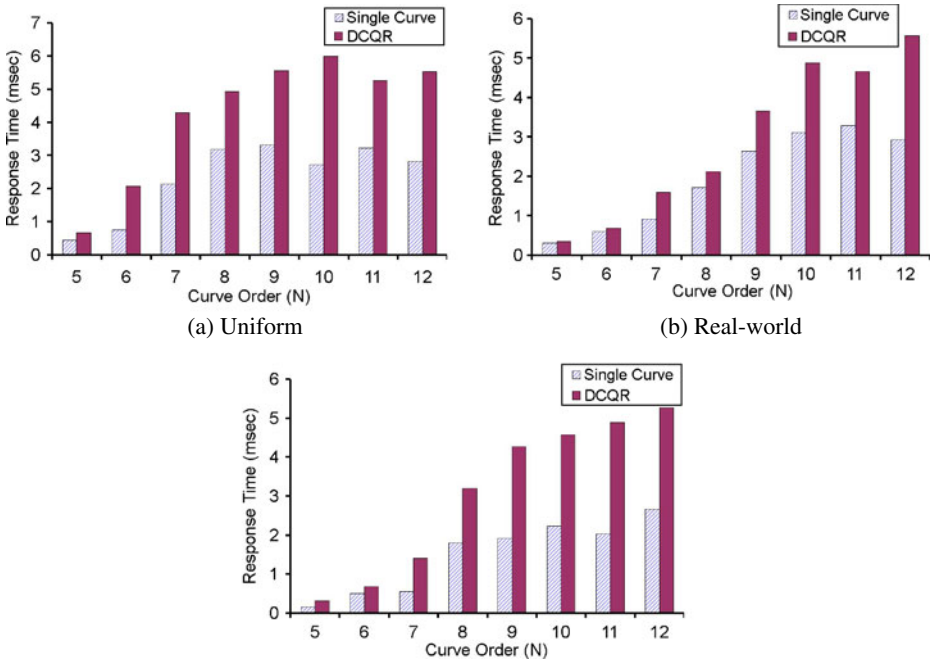
Fig. 14 Displacement vs. curve order (N)

derivation of Section 5. The overhead caused by DCQR is less than 6 milliseconds on average for all three datasets.

10.4 Range query evaluation

We now study the performance of our range query algorithm with the single curve and DCQR approaches. In contrast to the above experiments, our range query evaluation is exact. However, the query result set might include some excessive objects that should be filtered out by the client. Clearly, it is desirable to minimize the number of these excessive objects. Here, we first demonstrate the effect of Hilbert curve order (N) and the range query size on the number of excessive objects. We use precision (i.e., the percentage of objects falling in the range out of all objects retrieved) to quantify the amount of excessive objects. Figure 19 shows the effect of Hilbert curve order on the value of precision for our real-world dataset. This effect is evaluated for four different square range queries with the selectivity of 0.05 %, 0.1 % and 0.5 %. Clearly increasing the value of curve order results in higher precision values (reaching almost 1 for $N = 12$). Very similar trends were observed for the other two datasets.

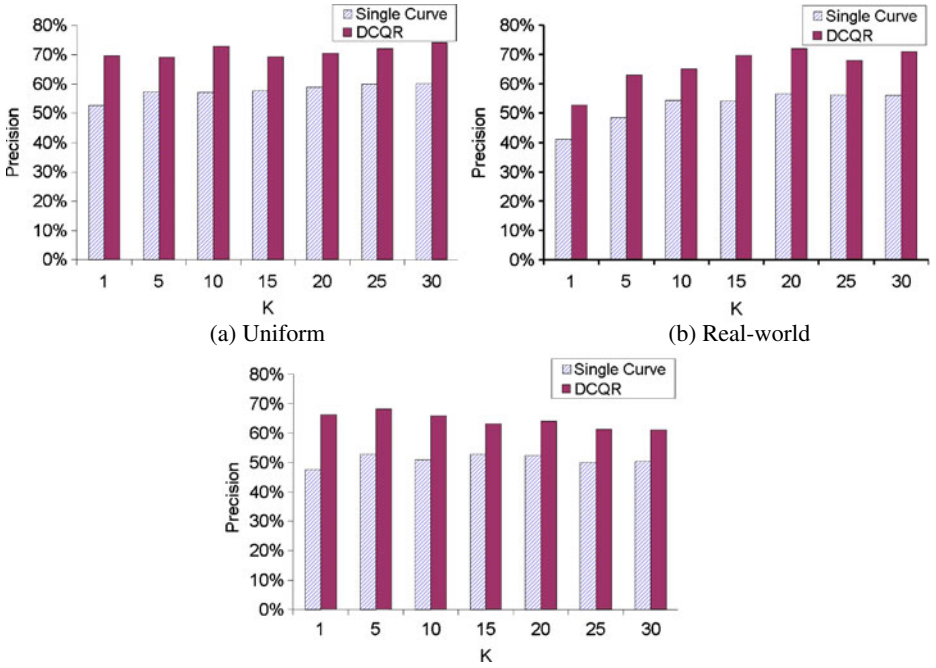
The second observation is that for a given N, larger range queries achieve better precision. This is because the excessive objects can only be in the grid cells which partially overlap with the range query. Clearly, the number of such cells is proportional to the perimeter of the range query. On the other hand, the number of



(a) Uniform (b) Real-world

(c) Skewed

Fig. 15 Response time vs. curve order (N)



(a) Uniform (b) Real-world

(c) Skewed

Fig. 16 Precision vs. K

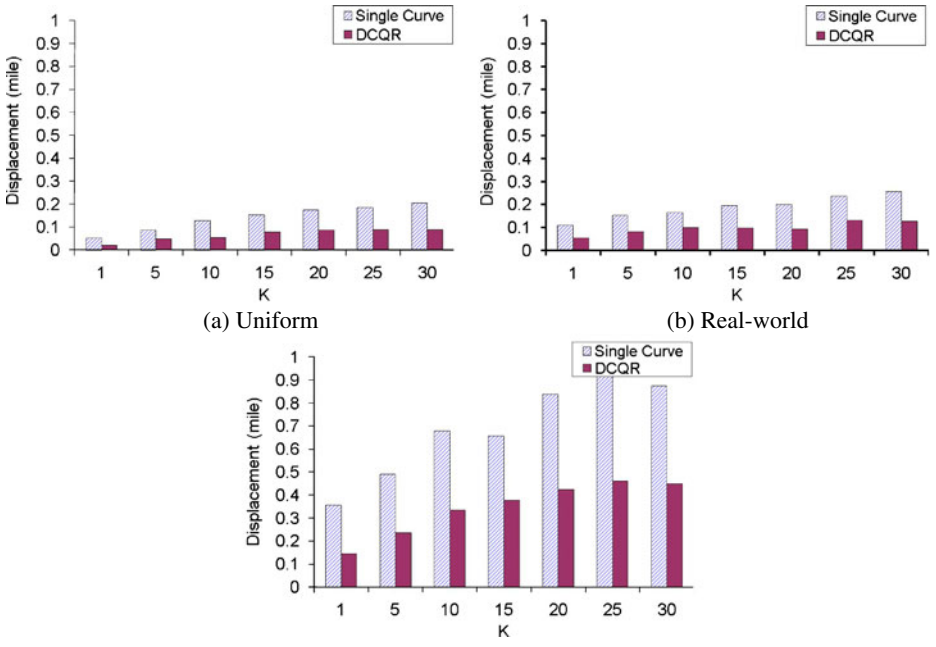


Fig. 17 Displacement vs. K

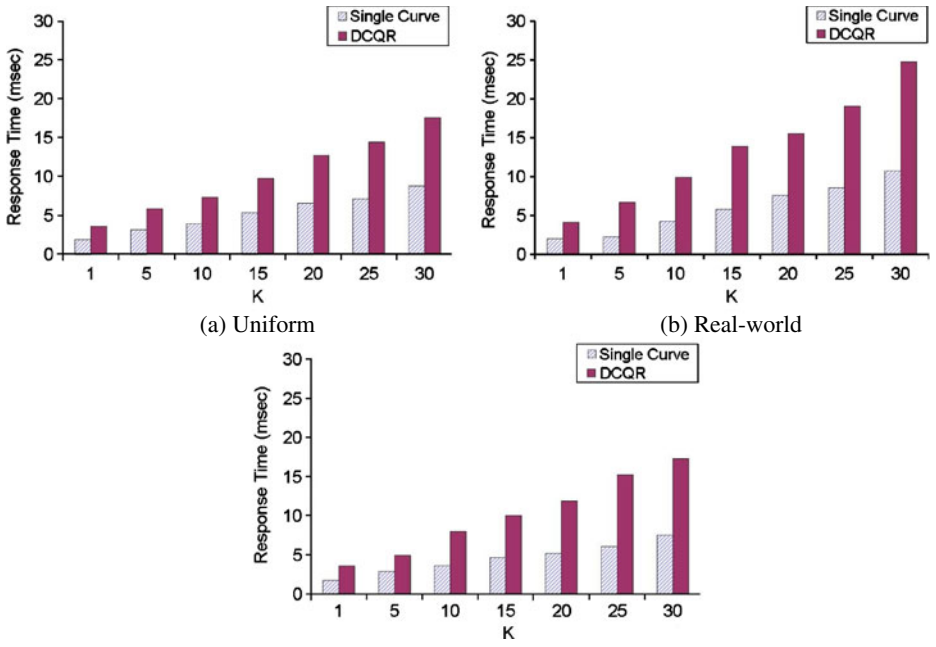


Fig. 18 Response time vs. K

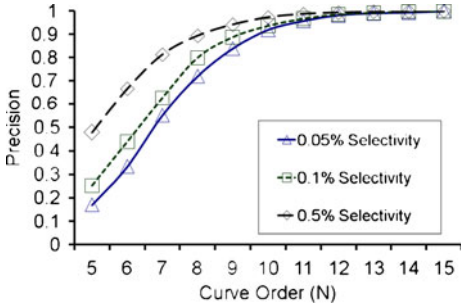
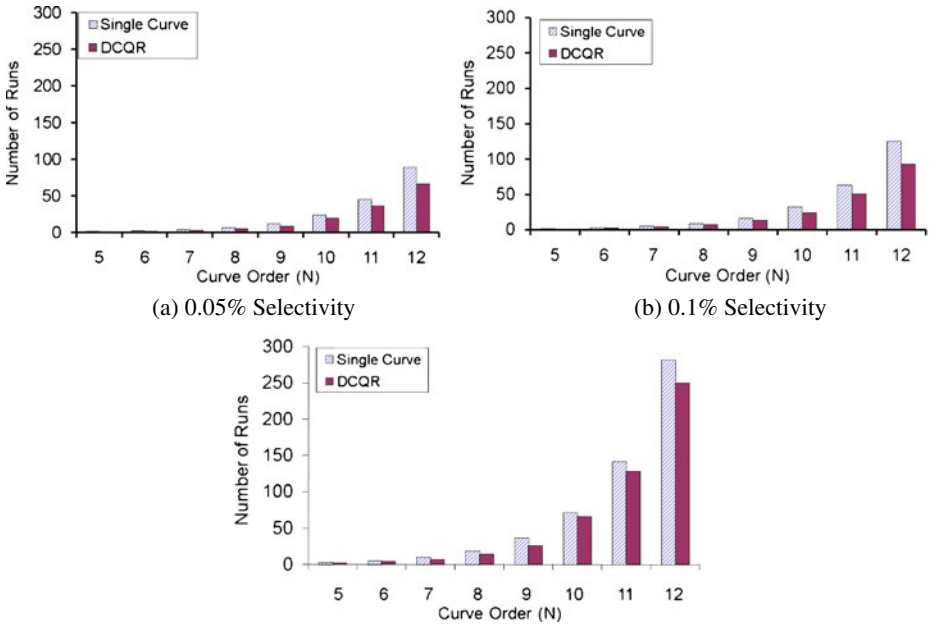


Fig. 19 Precision vs. curve order for different range query sizes

objects retrieved from the location server is on average proportional to the area of the query. Therefore, increasing the query side length increases the total number of retrieved objects more than the number of excessive objects.

As our second set of experiments we evaluate the effect of Hilbert curve order on the average number of query runs. It is desirable to have a small number of runs to reduce the communication cost as well as reducing the number of requests to the server (and hence increasing the server throughput). Figure 20 shows the average number of runs for 1,000 range queries over the real-world dataset with the selectivity of 0.05 %, 0.1 % and 0.5 %. For both original and DCQR approaches, as N increases, the average number of query runs increases as well. This is due to the fact that the average number of query runs is linearly proportional with the query side length in terms of the number of grid cells. Therefore, there is a trade-off in choosing the



(a) 0.05% Selectivity

(b) 0.1% Selectivity

(c) 0.5% Selectivity

Fig. 20 Number of runs vs. Hilbert order for a query with different selectivity

right curve order to minimize the number of query runs while maximizing precision. Considering different curve orders for a specific query size in Fig. 20, we get an average improvement of around 21 % when we use DCQR approach over original curve. Moreover, it is clear that larger range queries have more runs. Note that the average number of runs is independent from the dataset type and only depends on the coordinates of the query.

In the last set of our range query experiments we study the running time of the proposed range query algorithm. Figure 21a–d shows the running time of both original and DCQR approaches for different Hilbert curve orders for our real dataset with four different query sizes. For a fixed query side length, increasing the value of curve order increases the running time. This is consistent with the range query complexity derived in Section 6.1. Also, fortunately the overhead of DCQR approach is marginal compared to the original approach (around 6 milliseconds on average). This is important since it confirms the fact that we can use DCQR to evaluate a range query without significant time overhead. Note that both approaches retrieve the same number of objects from the location server and the difference between DCQR and original approach is the step required to find the runs in the dual curve. As the grid becomes more fine-grained, the time to retrieve objects from the location server dominates the time to calculate query runs and hence increases the overhead of DCQR. Similar trends were observed for the other two datasets. Although the distribution of objects varies for different datasets, the number of runs and the average number of objects retrieved for a unique query size are independent

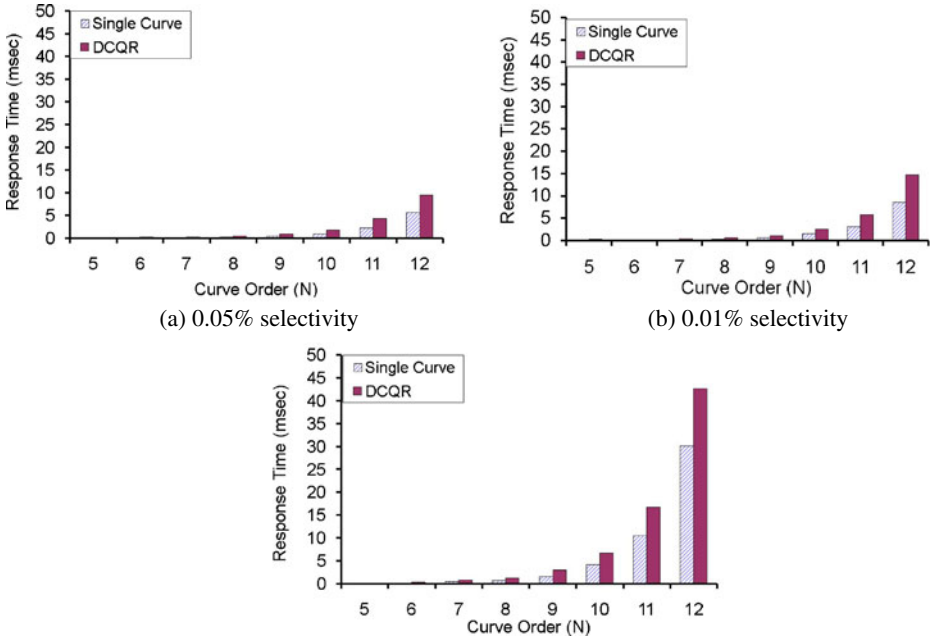


Fig. 21 Running times vs. curve order (N) for different query selectivity

from the dataset. Therefore, the running time is almost independent from the data distribution.

10.5 Exact KNN query evaluation

In the previous sections, we empirically evaluated the performance of approximate *K*NN and range algorithms and observed how $DCQR$ improves the precision and displacement for *K*NN queries and reduces the number of runs for range queries. In this section, we use these constructs with $DCQR$ to experimentally measure the performance of our exact *K*NN algorithm from Section 7.

Similar to Section 10.3, we first study the effect of the curve order N on the quality of results. Since our exact *K*NN algorithm employs the range algorithm for retrieving all objects in a square region centered at the query point, the server's response for this range request includes several objects that do not belong to the actual *K*NN result set. In our next experiment, we measure how the curve order affects the number of such excessive objects that are returned to the client. As Fig. 22 suggests, the average number of such objects quickly drops as N grows and approximately stays constant for larger values of N (where ρ approaches 1). Moreover, relatively more unnecessary objects are returned for the skewed dataset. This is obviously due to the fact that the average size of the circle centered at q is larger for skewed datasets as the approximate *K*NN algorithm has to expand beyond larger gaps in the Hilbert space to obtain the approximate result set.

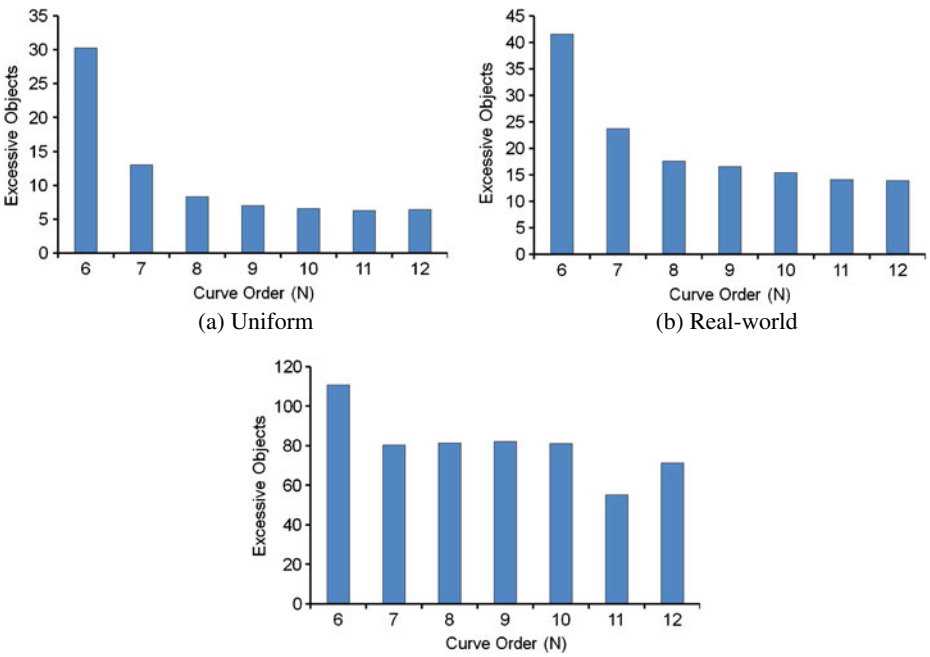


Fig. 22 Excessive objects vs. curve order (N)

We proceed to measure the overall response time for the exact *K*NN algorithm. The results are presented in Fig. 23 where contributions of each phase for query processing are highlighted. The *Aprx* piece denotes the time it takes to find the circular region that includes the *K* approximate results and the *Exact* piece represents the time it takes to retrieve all objects within the square range that encompasses the above region to compute the exact results. Several observations can be made from Fig. 23. The first phase of exact *K*NN query processing takes significantly less portion of the total running time for the algorithm. Moreover, as *N* grows, both *Aprx* and *Exact* portions of the response time grow and so does the total query time. This is consistent with the observations from Figs. 15 and 21. However, it stays below 20 milliseconds for our real-world and uniform datasets and below 120 milliseconds even for the skewed dataset.

As our last set of experiments, we evaluate the effect of increasing *K* on the number of excessive objects and the total time of our exact *K*NN algorithm. The results are shown in Figs. 24 and 25, respectively. As expected, increasing *K* almost linearly increases the number of excessive objects returned to clients as larger regions need to be queried with the range algorithm to find the exact results. The second observation from Fig. 24 is how the number of excessive objects for the skewed dataset are significantly larger than that of real-world and uniform datasets. This can be explained by revisiting Fig. 17. The skewness of data results in more displacement errors in finding the candidate *K*NN results that in turn leads to a relatively larger region to query with the range algorithm.

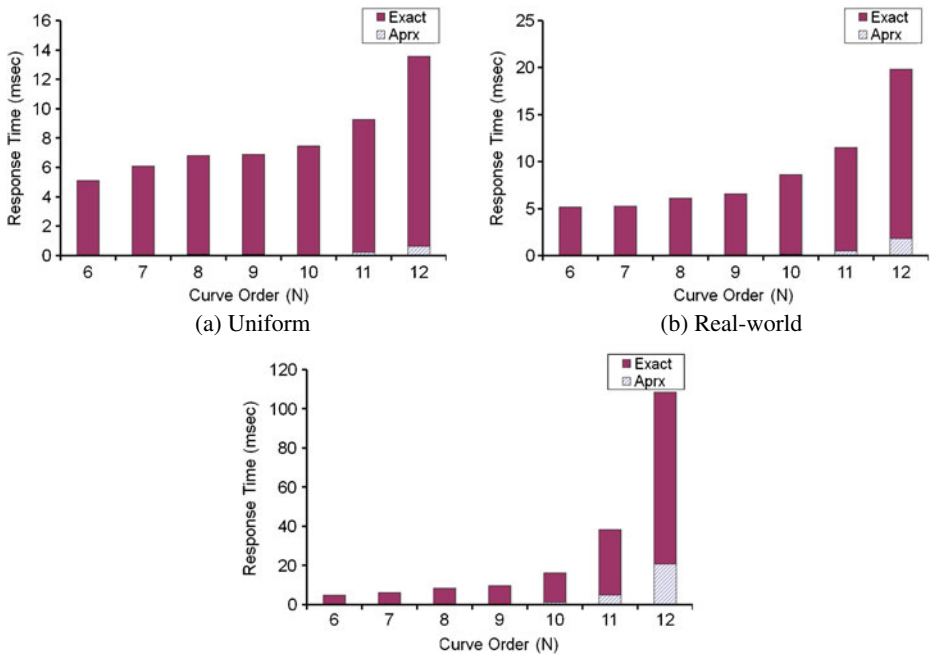


Fig. 23 Response time vs. curve order (N)

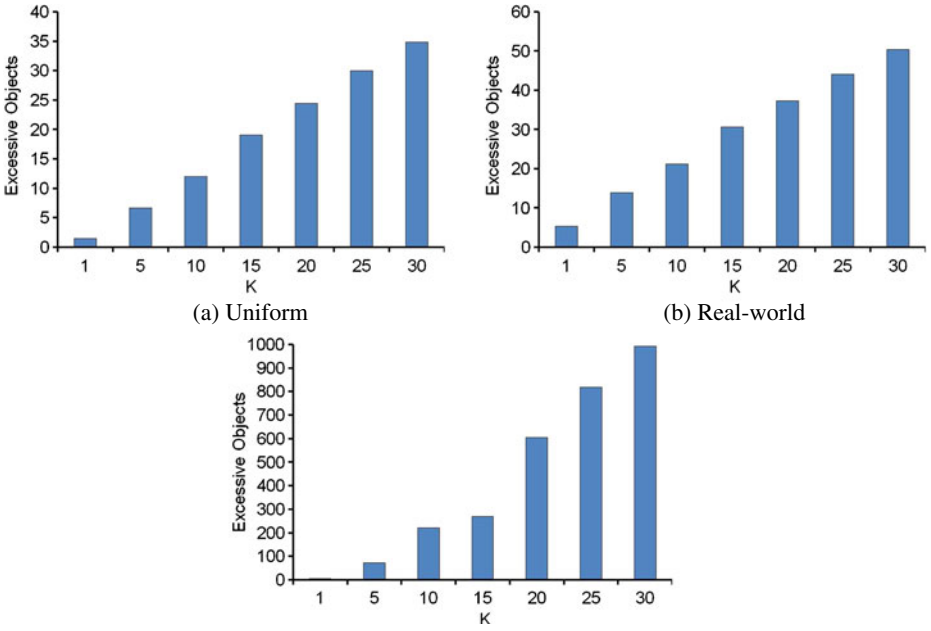


Fig. 24 KNN excessive objects vs. K

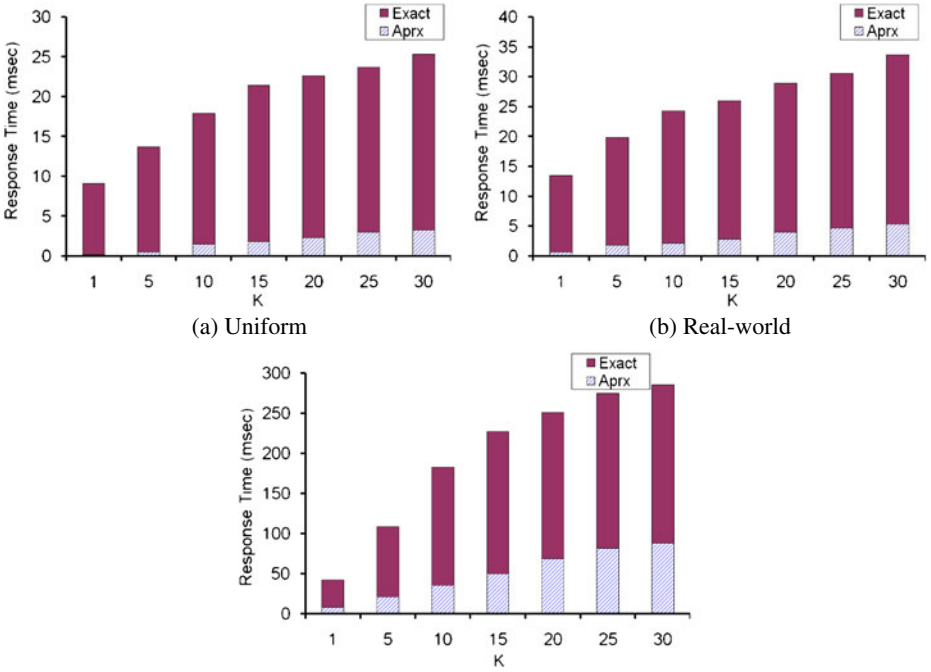


Fig. 25 KNN response time vs. K

With regard to overall response time, as Fig. 25 illustrates how increasing K results in an increase in overall response time. As we illustrated in Fig. 18 larger values of K result in more time spent to find the candidate result set and in a larger region. Querying this relatively larger region is the cause of an increase in the time contributions for the second phase of exact KNN query processing.

11 Conclusion and future work

In this paper, we discussed the problem of location privacy in location-based services. We studied the challenges of achieving location privacy and introduced a novel way of blindly evaluating spatial queries by using one-way space transformations to map objects and query points into an unknown space and addressing the query in this new space. The major contributions of our work can be summarized as follows: (i) We proposed blind evaluation of queries using Hilbert curves as space encoders and introduced *DCQR*, our proposed Dual Curve Query Resolution approach and showed how it improves the performance of blind range and KNN queries. (ii) We showed how our proposed space transformation employs space filling curves and cryptographic one-way hash functions to achieve stronger and more generalized measures of privacy than that of commonly used K-anonymity and spatial cloaking techniques without incurring the prohibitive costs of private information retrieval-based approaches. (iii) We conducted extensive experiments on our range, approximate and exact KNN algorithms using real-world and synthetic datasets to evaluate the performance of our proposed scheme.

As part of our future work, we intend to explore new privacy aware locality preserving space mappings that provide less overhead and allow blind evaluation of a wider class of spatial queries. Finally, in this paper we focused on the Euclidean space, and thus we plan to extend our approach to network-based spatial queries.

References

1. Asonov D (2004) Querying databases privately: a new approach to private information retrieval. Lecture notes in computer science, vol 3128. Springer
2. Beresford AR, Stajano F (2003) Location privacy in pervasive computing. IEEE Pervasive Computing 2(1):46–55
3. Bouganim L, Pucheral P (2002) Chip-secured data access: confidential data on untrusted servers. In: VLDB'02, pp 131–142
4. Chien H-Y, Jan J-K, Tseng Y-M (2002) An efficient and practical solution to remote authentication: smart card. Comput Secur 21(4):372–375
5. Chor B, Kushilevitz E, Goldreich O, Sudan M (1998) Private information retrieval. J ACM 45(6):965–981
6. Chung K-L, Tsai Y-H, Hu F-C (2000) Space-filling approach for fast window query on compressed images. IEEE Trans Image Process 9(12):2109–2116
7. Dingledine R, Mathewson N, Syverson PF (2004) Tor: the second-generation onion router. In: USENIX'04, pp 303–320
8. Faloutsos C, Jagadish H, Manolopoulos Y (1997) Analysis of the n-dimensional quadtree decomposition for arbitrary hyperectangles. IEEE Trans Knowl Data Eng 9(3):373–383
9. Faloutsos C, Roseman S (1989) Fractals for secondary key retrieval. In: PODS '89: proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. New York, NY, USA, pp 247–252

10. Gedik B, Liu L (2005) A customizable k-anonymity model for protecting location privacy. In: International conference on distributed computing systems (ICDPS), Columbos, Ohio
11. Ghinita G, Kalnis P, Khoshgozaran A, Shahabi C, Tan K-L (2008) Private queries in location based services: anonymizers are not necessary. In: SIGMOD'08, Vancouver, Canada
12. Gruteser M, Grunwald D (2003) Anonymous usage of location-based services through spatial and temporal cloaking. In: MobiSys. USENIX
13. Hilbert D (1891) Uber die stetige abbildung einer linie auf ein flachenstuck. Math Ann 38:459–460
14. Indyk P, Woodruff DP (2006) Polylogarithmic private approximations and efficient matching. In: Theory of cryptography, third theory of cryptography conference. New York, NY, USA, pp 245–264
15. Jagadish HV (1990) Linear clustering of objects with multiple atributes. In: Proceedings of the 1990 ACM SIGMOD international conference on management of data. ACM Press, Atlantic City, NJ, pp 332–342
16. Jagadish HV (1997) Analysis of the Hilbert curve for representing two-dimensional space. Inf Process Lett 62(1):17–22
17. Kalnis P, Ghinita G, Mouratidis K, Papadias D (2006) Preserving anonymity in location based services. A Technical Report
18. Khoshgozaran A, Shahabi C (2007) Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In: Advances in spatial and temporal databases, 10th international symposium, SSTD'07, 16–18 July, vol 4605. Boston, MA, USA, pp 239–257
19. Khoshgozaran A, Shahabi C, Shirani-Mehr H (2011) Location privacy: going beyond k-anonymity, cloaking and anonymizers. Knowl Inf Syst 26(3):435–465
20. Kushilevitz E, Ostrovsky R (1997) Replication is not needed: single database, computationally-private information retrieval. In: FOCS'97, pp 364–373
21. Lawder JK, King PJH (2001) Querying multi-dimensional data indexed using the Hilbert space-filling curve. SIGMOD Rec 30(1):19–24
22. Mokbel MF, Chow C-Y, Aref WG (2006) The new casper: query processing for location services without compromising privacy. In: Proceedings of the 32nd international conference on very large data bases. Korea, pp 763–774
23. Moon B, Jagadish HV, Faloutsos C, Saltz JH (2001) Analysis of the clustering properties of the Hilbert space-filling curve. IEEE Trans Knowl Data Eng 13(1):124–141
24. Preneel B (2003) Analysis and design of cryptographic hash functions. PhD thesis
25. Sagan H (1994) Space-filling curves. Springer
26. Samarati P, Sweeney L (1998) Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory
27. Tsai Y-H, Chung K-L, Chen W-Y (2004) A strip-splitting-based optimal algorithm for decomposing a query window into maximal quadtree blocks. IEEE Trans Knowl Data Eng 16(4):519–523

Ali Khoshgozaran received a BS degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2003, an MS degree in computer science from The George Washington

University, Washington DC, in 2005 and a PhD degree in computer science from University of Southern California. His research interests are in the areas of geospatial databases, locationbased services and location privacy.

Houtan Shirani-Mehr is currently a PhD student in Computer Science at the University of Southern California, Information Laboratory. He received his BS degree in Computer Engineering from Sharif University of Technology in 2004 and his MS degree in Information and Computer Science from University of California, Irvine in 2007. His research interests include planning, security and privacy and data mining.

Cyrus Shahabi is currently a Professor and the Director of the Information Laboratory (InfoLAB) at the Computer Science Department and also a Research Area Director at the NSF's Integrated Media Systems Center (IMSC) at the University of Southern California. He has two books and more than hundred research papers in the areas of databases, GIS and multimedia. He is currently on the editorial board of VLDB Journal, IEEE Transactions on Parallel and Distributed Systems and Journal of Spatial Information Science. He is the founding chair of IEEE NetDB workshop and also the general co-chair of ACM GIS 2007, 2008 and 2009. He regularly serves on the program committee of major conferences such as VLDB, SIGMOD, ICDE, SIGKDD, and Multimedia. Dr. Shahabi is the recipient of the US Presidential Early Career Awards for Scientists and Engineers (PECASE). He is a distinguished member of ACM and a senior member of IEEE.