Private Queries in Location Based Services: Anonymizers are not Necessary

Gabriel Ghinita[1], Panos Kalnis[1], Ali Khoshgozaran[2], Cyrus Shahabi[2], Kian-Lee Tan[1]

[1]Dept. of Computer Science
National University of Singapore
{ghinitag, kalnis, tankl}@comp.nus.edu.sg

[2]Dept. of Computer Science
University of Southern California
{jafkhosh, shahabi}@usc.edu*

ABSTRACT

Mobile devices equipped with positioning capabilities (e.g., GPS) can ask location-dependent queries to Location Based Services (*LBS*). To protect privacy, the user location must not be disclosed. Existing solutions utilize a trusted anonymizer between the users and the LBS. This approach has several drawbacks: *(i)* All users must trust the third party anonymizer, which is a single point of attack. *(ii)* A large number of cooperating, trustworthy users is needed. *(iii)* Privacy is guaranteed only for a single snapshot of user locations; users are not protected against correlation attacks (e.g., history of user movement).

We propose a novel framework to support private location-dependent queries, based on the theoretical work on Private Information Retrieval (*PIR*). Our framework does not require a trusted third party, since privacy is achieved via cryptographic techniques. Compared to existing work, our approach achieves stronger privacy for snapshots of user locations; moreover, it is the first to provide provable privacy guarantees against correlation attacks. We use our framework to implement approximate and exact algorithms for nearest-neighbor search. We optimize query execution by employing data mining techniques, which identify redundant computations. Contrary to common belief, the experimental results suggest that PIR approaches incur reasonable overhead and are applicable in practice.

Categories and Subject Descriptors

H.2.0 [**General**]: Security, integrity, and protection; H.2.8 [**Database applications**]: Spatial databases and GIS

General Terms

Design, Experimentation, Security

Keywords

Location Anonymity, Private Information Retrieval, Query Privacy

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.

1. INTRODUCTION

An increasing number of communication devices (e.g., mobile phones, PDAs), feature positioning capabilities (e.g., GPS). Users can ask location-dependent queries, such as "find the nearest hospital", which are answered by Location Based Services (*LBS*) like Mapquest or Google Maps. However, queries may disclose sensitive information about individuals, including health condition, lifestyle habits, political and religious affiliations, or may result in unsolicited advertisement (i.e., spam). Privacy concerns are expected to rise as LBSs become more common.

Observe that privacy is not protected by replacing the real user identity with a fake one (i.e., pseudonym), because, in order to process location-dependent queries, the LBS needs the exact location of the querying user. An attacker, which may be the LBS itself, can infer the identity of the query source by associating the location with a particular individual. This can be easily performed in practice, with the help of a public telephone directory, for instance, which contains subscribers' addresses.

Most existing solutions adopt the *K-anonymity* [22] principle: a query is considered private, if the probability of identifying the querying user does not exceed $1/K$, where K is a user-specified anonymity requirement. To enforce this principle, a trusted third-party *anonymizer* is employed [11, 14, 17, 20] (see Figure 1). The anonymizer maintains the current locations of all subscribed users. Instead of sending the Nearest Neighbor (*NN*) query to the LBS, the user (Alice in our example) contacts the anonymizer, which generates a Cloaking Region (*CR*) enclosing Alice as well as $K-1$ other users in her vicinity. In Figure 1, $K=3$ and the CR contains u_1 and u_2 in addition to Alice. The CR is sent to the LBS, which cannot identify Alice with probability larger than $1/K$. The LBS computes a candidate set that includes all points of interest (*POI*) which may potentially be the NN for any point within the entire CR [15]. The candidate set (i.e., $\{p_1, p_2, p_3, p_4\}$) is sent back to the anonymizer, which filters the false hits and returns the actual NN (i.e., p_3) to Alice. We discuss these methods in Section 2.

Existing methods have several drawbacks: *(i)* The anonymizer is a single point of attack: if an attacker gains access to it, the privacy of all users is compromised. It is also a bottleneck, since it must process the frequent updates of user locations. *(ii)* A large number of users must subscribe to the service, otherwise CR cannot be constructed. It is assumed that all users are trustworthy. However, if some of them are malicious, they can easily collude to compromise the privacy

*Authors' work has been funded by NSF grant NSF-0742811

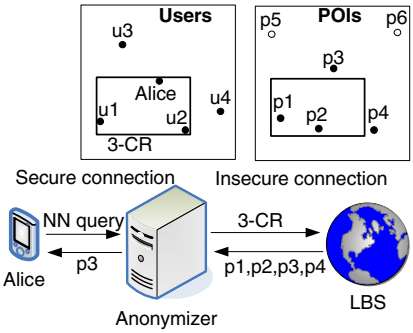

Figure 1: Existing three-tier architecture

of a targeted user. *(iii)* It is assumed that the attacker has no background information about the users, but in practice it is difficult to model the exact knowledge. Assume that Alice is searching for the nearest women's clinic, and the CR contains Alice and Bob. From the query content, the attacker can identify Alice as the query source. *(iv)* Privacy is guaranteed only within a static snapshot of user locations; users are not protected against correlation attacks. For example, if Alice asks the same query from different locations as she moves, she can be easily identified because she will be included in all CRs.

We propose a framework for private location-dependent queries that solves these problems. Our framework is based on the theory of Private Information Retrieval (*PIR*) and does *not* need an anonymizer. Recent research on PIR [4, 19] resulted in protocols that allow a client to privately retrieve information from a database, without the database server learning what particular information the client has requested. Most techniques are expressed in a theoretical setting, where the database is an n-bit binary string X (see Figure 2). The client wants to find the value of the i^{th} bit of X (i.e., X_i). To preserve privacy, the client sends an encrypted request $q(i)$ to the server. The server responds with a value $r(X, q(i))$, which allows the client to compute X_i. We focus on *computational* PIR, which employs cryptographic techniques, and relies on the fact that it is computationally intractable for an attacker to find the value of i, given $q(i)$. Furthermore, the client can easily determine the value of X_i based on the server's response $r(X, q(i))$. PIR theory is discussed in Section 3.

In this paper we show that PIR can be used to compute privately the nearest neighbor of a user with acceptable cost, by retrieving a small fraction of the LBS' database. Consider the example of Figure 3.a, where u is the querying user and the LBS contains four points of interest p_1, p_2, p_3, p_4. In an off-line phase, the LBS generates a kd-tree index of the POIs and partitions the space into three regions A, B, C. To answer a query, the server first sends to u the regions

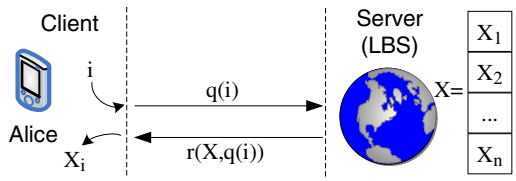

Figure 2: PIR framework

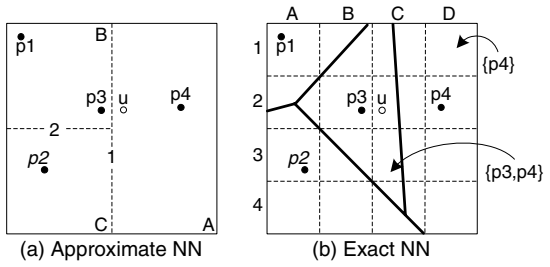

Figure 3: Finding the Nearest Neighbor of u

A, B, C. The user finds the region (i.e., A) that contains him, and utilizes PIR to request all points within A; therefore, the server does not know which region was retrieved. The user receives the POIs in A in encrypted form and calculates p_4 as his NN. The method can be used with a variety of indices. In Section 4 we present implementations based on the Hilbert curve and on an R-tree variant. Note that the result is approximate; in our example the true NN is p_3. We show experimentally that the approximation error is low.

We also propose a method for finding the exact NN. In a pre-processing phase, the server computes the Voronoi diagram for the POIs (see Figure 3.b). Each POI p_i is assigned to its Voronoi cell; by definition, p_i is the NN of any point within that cell. The server superimposes a regular grid of arbitrary granularity on top of the Voronoi diagram. Each grid cell stores information about the Voronoi cells intersecting it. For example $D1$ stores $\{p_4\}$, whereas $C3$ stores $\{p_3, p_4\}$. Upon asking a query, the client first retrieves the granularity of the grid, and calculates the grid cell that contains him (i.e., $C2$). Then, he employs PIR to request the contents of $C2$. He receives $\{p_3, p_4\}$ (encrypted) and calculates p_3 as his exact NN. The method is described in Section 5. Note that the cost is typically higher compared to approximate NN.

PIR has been criticized of being too costly to be applied in practice. [24] showed that the computational time of PIR may be longer than the time required for an oblivious transfer of the database. [24] assumes that the server agrees to surrender the entire database to the client. In practice, this is rarely the case, since the database is a valuable asset for the server, who charges the client, either directly or indirectly (e.g., advertisements), based on the actual usage. Moreover, most queries do not need to retrieve the entire database. Still, the CPU cost of PIR can be high, since it involves numerous multiplications of large numbers. Nevertheless, we show that much of the computation is redundant. In Section 6 we develop a query optimizer which employs data mining techniques to identify such redundancy. The resulting execution plan is up to 40% cheaper in terms of CPU cost. Moreover, we show that the required computations are easily parallelized.

Summarizing, our contributions are:

1. We propose a novel framework for private location-dependent queries, which uses PIR protocols and eliminates the need for any trusted third party. Our work is the first to provide provable privacy guarantees against correlation attacks.

2. We develop algorithms for approximate and exact private nearest neighbor search. We utilize data mining techniques to optimize query execution.

3. We show experimentally that the cost is reasonable; hence our methods are applicable in practice.

2. RELATED WORK

Most existing approaches for private location-dependent queries follow the user-anonymizer-LBS framework of Figure 1. The anonymizer sends to the LBS a Cloaking Region (CR) instead of the actual user location; this procedure is called *cloaking*. Ref. [11] combines spatial with temporal cloaking. Each query q specifies a temporal interval δt that the corresponding user u is willing to wait. If within δt, $K-1$ other clients in the vicinity of u also issue queries, all these queries are combined in a single CR; otherwise, q is rejected. In *Casper* [20], the anonymizer maintains the locations of the clients using a pyramid data structure, similar to a Quad-tree. Assume u asks a query and let c be the lowest-level cell of the Quad-tree where u lies. If c contains enough users (i.e., $|c| \geq K$), c becomes the CR. Otherwise, the horizontal c_h and vertical c_v neighbors of c are retrieved. If $|c \cup c_h| \geq K$ or $|c \cup c_v| \geq K$, the corresponding union of cells becomes the CR; otherwise, the anonymizer retrieves the parent of c and repeats this process recursively. *Interval Cloak* [14] is similar to Casper in terms of both the data structure used by the anonymizer (a Quad-tree), and the cloaking algorithm. The main difference is that Interval Cloak does not consider neighboring cells at the same level when determining the CR, but ascends directly to the ancestor level. Casper and Interval Cloak guarantee privacy only for uniform distribution of user locations.

Hilbert Cloak [17] uses the Hilbert space filling curve to map the 2-D space into 1-D values. These values are then indexed by an annotated B^+-tree. The algorithm partitions the 1-D sorted list into groups of K users (the last group may have up to $2K-1$ users). For querying user u the algorithm finds the group to which u belongs, and returns the minimum bounding rectangle of the group as the CR. The same CR is returned for any user in a given group. Hilbert Cloak guarantees privacy for any distribution of user locations.

The previous approaches assume a static snapshot of user locations and do not consider correlation attacks (e.g., history of user movement). In [5], correlation attacks are handled as follows: At the initial timestamp t_0, cloaking region CR_0 is generated, which encloses a set AS of at least K users. At a subsequent timestamp t_i, the algorithm computes a new anonymizing region CR_i that encloses the same users in AS, but contains their locations at timestamp t_i. There are two drawbacks: *(i)* As users move, the resulting CR may grow very large, leading to prohibitive query cost. *(ii)* If a user in AS disconnects from the service, the query must be dropped. Furthermore, in [5] it is assumed that there are no malicious users.

Privacy in LBS has also been studied in the context of related problems. In [3], the CR is a closed region around the query point, which is independent of the number of users inside. Given CR, the LBS returns the probability that each candidate result satisfies the query based on its location with respect to the CR. Ref. [6], on the other hand, eliminates the anonymizer by organizing users in a Peer-to-Peer system. The querying user u searches his neighborhood until he finds $K-1$ other peers, which are used to construct the CR. However, u tends to be close to the center of the CR; therefore an attacker can identify u with high probability. Ref. [12] also uses a Peer-to-Peer system to support distributed anonymization; although a centralized anonymizer is not required, all users must trust each other. More relevant to our work is the approach of [18]: In a preprocessing phase, a trusted third party transforms (using 2-D to 1-D mapping) and encrypts the database. The database is then uploaded to the LBS, which does not know the decryption key. All users possess tamper-resistant devices which store the decryption key, but they do not know the key themselves. Users send encrypted queries to the LBS and decrypt the answers to extract the results. The method assumes that none of the tamper-resistant devices is compromised. If this condition is violated, the privacy of all users is threatened.

Our work builds on the theoretical results of Private Information Retrieval (PIR), which is defined as follows: a server S holds a database with n bits, $X = (X_1 \dots X_n)$. A user u has a particular index i and wishes to retrieve the value of X_i, without disclosing to S the value of i. The PIR concept was introduced in [4] in an information theoretic setting, requiring that even if S had infinite computational power, it could not find i. It is proven in [4] that in any solution with a single server, u must receive the entire database (i.e., $\Theta(n)$ cost). Nevertheless, in practice, it is sufficient to ensure that S cannot find i with polynomial-time computations; this problem is known as *Computational PIR*. [19] showed that the communication cost for a single server is $\Theta(n^{\varepsilon})$, where ε is an arbitrarily small positive constant. Our work employs Computational PIR.

Several approaches employ cryptographic techniques to privately answer NN queries over relational data. Most of them are based on some version of the secure multiparty computation problem [13]. Let two parties A and B hold objects a and b, respectively. They want to compute a function $f(a, b)$ without A learning anything about b and vice versa. They encrypt their objects using random keys and follow a protocol, which results into two "shares" S_A and S_B given to A and B, respectively. By combining their shares, they compute the value of f. In contrast to our problem (which hides the querying user from the LBS), existing NN techniques assume that the query is public, whereas the database is partitioned into several servers, neither of which wants to reveal their data to the others. Ref. [25] assumes vertically partitioned data and uses secure multiparty computation to implement a private version of Fagin's [8] algorithm. Ref. [23] follows a similar approach, but data is horizontally partitioned among the servers. The computation cost is $O(n^2)$ and may be prohibitive in practice. Ref. [1] also assumes horizontally partitioned data, but focuses on top-k queries.

More relevant to our problem is the work of [16] which uses PIR to compute the NN of a query point. The server does not learn the query point and the user does not learn anything more than the NN. To achieve this, the method computes private approximations of the Euclidean distance by adapting an algorithm [9] that approximates the Hamming distance in $\{0,1\}^d$ space (d is the dimensionality). The cost of [16] is $\tilde{O}(n^2)$ for the exact NN and $\tilde{O}(\sqrt{n})$ for an approximation through sampling. The paper is mostly of theoretical interest, since the $\tilde{O}$ notation hides polylogarithmic factors that may affect the cost; the authors do not provide any experimental evaluation of the algorithms.

Symbol	Description
k	Modulus Bits
q_1, q_2	$k/2$-bit primes
$N = q_1 \cdot q_2$	Modulus
n	Number of Data Objects
m	Object Size (bits)
$t = \lceil \sqrt{n} \rceil$	PIR Matrix Dimension
$M_{1:t,1:t}[1:m]$	PIR Matrix (binary array)
$y_{1:t}$, array of k-bit numbers	PIR Query
$z_{1:t}[1:m]$, array of k-bit numbers	PIR Reply

Table 1: Summary of notations

3. PIR FRAMEWORK FOR LBS

This section provides an overview of the proposed PIR framework: Section 3.1 outlines an existing PIR protocol for binary data, which we use as a building block in our techniques. Section 3.2 discusses the advantages of our PIR framework, compared to existing spatial cloaking techniques.

3.1 Computational PIR Protocol

Computational PIR [19] relies on the *Quadratic Residuosity Assumption (QRA)*, which states that it is computationally hard to find the quadratic residues in modulo arithmetic of a large composite number $N = q_1 \cdot q_2$, where q_1, q_2 are large primes (see Table 1 for a summary of notations).

Define

$$\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N | gcd(N, x) = 1\}, \quad (1)$$

the set of numbers in $\mathbb{Z}_N$ which are prime with N (gcd is the greatest common divisor). Then the set of *quadratic residues (QR)* modulo N is defined as:

$$QR = \{y \in \mathbb{Z}_N^* | \exists x \in \mathbb{Z}_N^* : y = x^2 \mod N\}. \quad (2)$$

The complement of QR with respect to $\mathbb{Z}_N^*$ constitutes the set of *quadratic non-residues (QNR)*.

Let

$$\mathbb{Z}_N^{+1} = \{y \in \mathbb{Z}_N^* | \left(\frac{y}{N}\right) = 1\}, \quad (3)$$

where $\left(\frac{y}{N}\right)$ denotes the Jacobi symbol [10]. Then, exactly half of the numbers in $\mathbb{Z}_N^{+1}$ are in QR, while the other half are in QNR. According to QRA, for $y \in \mathbb{Z}_N^{+1}$, it is computationally intractable to decide whether $y \in QR$ or $y \in QNR$. Formally, define the quadratic residuosity predicate Q_N such that:

$$Q_N(y) = 0 \Leftrightarrow y \in QR \quad (4)$$

Then, if q_1 and q_2 are $\frac{k}{2}$-bit primes, for every constant c and any function $C(y)$ computable in polynomial time, there exists k_0 such that

$$\forall k > k_0, \Pr_{y \in \mathbb{Z}_N^{+1}}[C(y) = Q_N(y)] < \frac{1}{2} + \frac{1}{k^c} \quad (5)$$

Hence, the probability of distinguishing between a QR and a QNR is negligible for large-enough k.

Let $t = \lceil \sqrt{n} \rceil$ and consider that the database X is organized as a square $t \times t$ matrix M (the matrix is padded with extra entries if n is not a perfect square). Let $M_{a,b}$ be the matrix element corresponding to X_i that is requested by the user u. u randomly generates modulus N (similar to a public key in asymmetric cryptography), and sends it to the server, together with query message $y = [y_1 \dots y_t]$, such that $y_b \in QNR$, and $\forall j \neq b, y_j \in QR$.

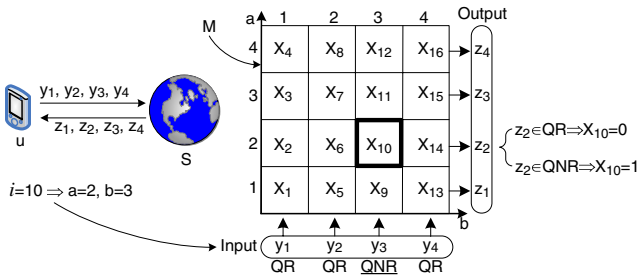

Figure 4: PIR example. u requests X_{10}

The server computes for every row r of M the value

$$z_r = \prod_{j=1}^{t} w_{r,j} \quad (6)$$

where $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j otherwise[1]. The server returns $z = [z_1 \dots z_t]$. Based on the *Euler criterion*, u computes the following formula:

$$\left(z_a^{\frac{q_1-1}{2}} = 1 \mod q_1\right) \wedge \left(z_a^{\frac{q_2-1}{2}} = 1 \mod q_2\right) \quad (7)$$

If Equation 7 is $true$, then $z_a \in QR$ else $z_a \in QNR$. Since u knows the factorization of N, Equation 7 can be efficiently computed using the Legendre symbol [10]. The user determines the value of $M_{a,b}$ as follows: If $z_a \in QR$ then $M_{a,b} = 0$, else $M_{a,b} = 1$.

EXAMPLE 1. Figure 4 shows an example, where $n = 16$. u requests X_{10}, which corresponds to $M_{2,3}$. Therefore, u generates a message $y = [y_1, y_2, y_3, y_4]$, where $y_1, y_2, y_4 \in QR$ and $y_3 \in QNR$. The server replies with the message $z = [z_1, z_2, z_3, z_4]$. If $z_2 \in QR$ then u concludes that $X_{10} = 0$, else $X_{10} = 1$.

The protocol requires $O(n)$ multiplications at the server, and $O(\sqrt{n})$ communication cost. The latter can be reduced to $O(n^\varepsilon)$, $0 < \varepsilon < 1/2$, by applying the method recursively [19]. Although the recursive variation is asymptotically better than the basic one, our experiments revealed that the overhead of the recursion is not justified in practice.

The previous protocol retrieves privately one bit of information. The same idea can be extended to retrieve an object p_i which is represented as an m-bit binary string. Let D be a database containing n objects: $D = \{p_1, p_2, \dots p_n\}$. Again, the server generates a matrix M with the difference that each cell contains an m-bit object. Conceptually, this is equivalent to maintaining m matrices $M[1], M[2], \dots M[m]$, one for each bit of the objects. Assume that u requests object p_i. Same as the 1-bit case, u sends a message $y = [y_1 \dots y_t]$. However, the server applies y to each one of the m matrices, resulting to m answer messages: $z[1], z[2], \dots z[m]$. u receives these messages and computes all m bits of p_i. The communication and computational cost increase to $O(m\sqrt{n})$ and $O(m \cdot n)$, respectively. In the rest of the paper, we use

$$PIR(p_i)$$

to denote that a user u retrieves privately an object p_i from the server, using the described protocol.

[1]According to [24] the formula can be simplified as follows: $w_{r,j} = y_j$ if $M_{r,j} = 1$, otherwise $w_{r,j} = 1$

3.2 Private Location-dependent Queries

There are two privacy issues in location-dependent queries: *(i)* The user must hide his identity (e.g., username, IP address, etc). This is orthogonal to our problem and can be achieved through a widely available anonymous web browsing service (that service does not learn the location of u). *(ii)* The user must hide his location. Similar to previous research on spatial K-anonymity (see Section 2), our PIR framework focuses on this issue. The advantages of our approach are:

PIR does not disclose *any* spatial information. As opposed to CR-based methods (which only perturb location, but still disclose the CR), no location information is disclosed. Instead, the data (i.e., POIs) are retrieved based on object index, by employing the provably private PIR protocol. This approach prevents any type of attack based on user location. In Sections 4 and 5, we develop methods to find the NN of a user with exactly one PIR request, irrespective of his location.

PIR protects against correlation attacks. Assume that u asks a continuous query as he moves. Existing methods generate one cloaking region CR_i per location, but all CR_i will include u. By intersecting the set of users in all CR_i, an attacker can identify u with high probability; this is called *correlation attack*. Note that this attack is possible because the CR reveals spatial information. Since the PIR framework does not reveal any spatial information, u is protected against correlation attacks.

PIR reduces significantly the identification probability. Let U be the set of all possible users (e.g., all mobile phone users within a country); $|U|$ is typically a large number (i.e., in the order of millions). From the server's point of view, the PIR request may have originated from any $u_i \in U$. Therefore, the probability to identify u as the querying user is $1/|U|$. In contrast, K-anonymity techniques require a subset of users $U' \subset U$ to subscribe to the anonymization service; typically $|U'| \ll |U|$. Moreover, only K users are included in a cloaking region CR, and $K \ll |U'|$, otherwise CR grows large and the query cost becomes prohibitive (typically K is in the order of 10^2 [17, 20]). Therefore, the probability $1/K$ of identifying u is several orders of magnitude larger than that of the PIR framework.

PIR does not require any trusted third party, since privacy is achieved through cryptographic techniques. Existing techniques, on the other hand, need: (i) An anonymizer, which is a single point of attack, and (ii) A large set U' of subscribed users, all of whom must be trustworthy, since malicious users may collude to reveal the location of u. Furthermore, users in U' must accept the cost of sending frequent location updates to the anonymizer, even if they do not ask queries.

PIR reduces the number of disclosed POI. Existing techniques disclose a large set of candidate POIs (see Figure 1). Since the database is a valuable asset of the LBS, users may be charged according to the result size. We will show (Section 7) that PIR techniques disclose far fewer POIs.

4. APPROXIMATE NEAREST NEIGHBORS

In this section we describe our *ApproxNN* method, which employs the PIR framework to retrieve privately the nearest point of interest (i.e., NN) of u from a LBS. We show that a good approximation of the NN can be found with only one PIR request. For simplicity, in Section 4.1 we describe our method using the 1-D Hilbert ordering. In Section 4.2 we generalize to 2-D partitionings, such as kd-trees and R-trees.

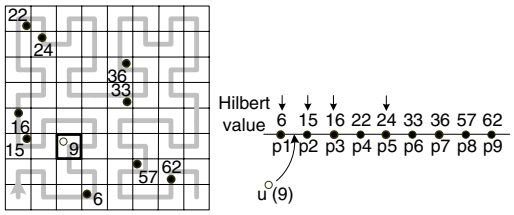

Figure 5: 9 POIs on a 8×8 Hilbert curve

4.1 Approximate NN using Hilbert ordering

The Hilbert space filling curve is a continuous fractal that maps the 2-D space to 1-D. Let p_i be a POI and denote its Hilbert value as $H(p_i)$. The Hilbert ordering of a database $D = \{p_1, p_2, \ldots p_n\}$ is a list of all objects sorted in ascending order of their Hilbert values. Figure 5 shows an example database with 9 POIs $D = \{p_1, \ldots p_9\}$, where $H(p_1) = 6$, $H(p_2) = 15$, etc. The granularity of the Hilbert curve is 8×8. The granularity does not affect the cost of our method, therefore it can be arbitrarily fine.

If two POIs are close in the 2-D space, they are likely to be close in the Hilbert ordering, as well [21]. Therefore, an approximation of the NN of u is the POI p_i whose Hilbert value $H(p_i)$ is closest to $H(u)$. Since the POIs are sorted on their Hilbert value, we can use binary search to compute the approximate NN in $O(\log n)$ steps. In our example, $H(u) = 9$, therefore we retrieve $p_5 \rightarrow p_3 \rightarrow p_2 \rightarrow p_1$. The answer is p_1 since its distance from u in the 1-D space is $|H(p_1) - H(u)| = |6 - 9| = 3$, which is the smallest among all POIs. Note that the answer is approximate; the true NN is p_2.

There are two problems with this approach: First, since the search must not reveal any information, $O(\log n)$ costly private requests for $PIR(p_i)$ must be performed. Second, a side effect of the PIR protocol is that each $PIR(p_i)$ retrieves not one, but $\sqrt{n}$ POIs. Recall the example of Figure 4, where u is interested in X_{10}. The server returns z_1, z_2, z_3, z_4, from which u can compute the entire column 3 of M, i.e., $X_9, X_{10}, X_{11}, X_{12}$. Consequently, the binary search will retrieve $O(\sqrt{n} \log n)$ POIs, which represent a large fraction of the database.

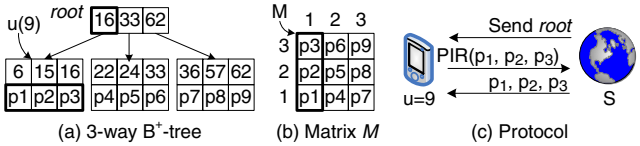

Figure 6: Approximate NN using Hilbert

Observe, however, that each PIR request is intuitively analogous to a "page access" on a disk. Therefore, the POIs can be arranged in a B$^+$-tree, where each node contains at most $\lceil \sqrt{n} \rceil$ POIs. The B$^+$-tree for our running example is shown in Figure 6.a; since there are 9 POIs, the capacity of each node is 3. Each entry in the root has a key and a pointer to a leaf. All Hilbert values in a leaf are less or equal

Approximate NN Protocol

User u: Initiate query
Server: Send root node
User u: Let b be the column that includes u
$y = [y_1 : y_{\sqrt{n}}]$, $y_b \in QNR$, and $\forall j \neq b, y_j \in QR$
Send y
Server: Send $z[1:m] = [z_1 : z_{\sqrt{n}}][1:m]$
User u: Calculate distance to all POIs in column b
Return the approximate NN

Figure 7: Protocol for approximate NN

to the corresponding root key. Each leaf node corresponds to one column of the PIR matrix M (see Figure 6.b). Note that M stores the POIs without their Hilbert value. Without loss of generality, we assume that each POI consists of its coordinates: $p_i = (x_i, y_i)$; more complex objects are easily supported. During query processing the server sends to u the root node (i.e., $\langle 16, 33, 62 \rangle$). In the example $H(u) = 9 \leq 16$, therefore u must retrieve privately the first column of M. This is done with one request $PIR(\{p_1, p_2, p_3\})$. Next, u computes his NN from the set $\{p_1, p_2, p_3\}$. The answer is p_2, which happens to be the exact NN. Note that by retrieving several POIs in the neighborhood of u, the approximation error decreases; however, the method remains approximate.

Observe that the height of the tree is always $\log_{\sqrt{n}} n = 2$. The fact that u asks for the root node does not reveal any information to the server, since all queries require the root. Therefore, the server sends the root, which contains only Hilbert values but no POIs, in a low-cost plain format (i.e., does not use PIR). Consequently, the NN is computed with only *one* PIR request (i.e., one column of M). Figure 7 shows the protocol. The communication cost is $O(\sqrt{n})$ and u retrieves up to $\sqrt{n}$ POIs; for instance, if the LBS contains 10^6 POIs, u retrieves 0.1% of them. In Section 7 we show that existing K-anonymity methods retrieve more POIs.

4.2 Generalization to 2-D partitionings

The previous method can be extended to 2-D partitionings. The only requirement is that data must be partitioned into at most $\sqrt{n}$ buckets, each containing up to $\sqrt{n}$ POIs. Consider the case of the *kd*-tree [7]. The original insertion algorithm partitions the space either horizontally or vertically such that every partition contains one point. We modify the algorithm as follows: Let n' be the number of POIs in the current partition (initially $n' = n$), and let g be the number of remaining available partitions (initially, there are $\sqrt{n}$). We allow splits that create partitions e_1 and e_2 such that $|e_1| + |e_2| = n'$ and

$$\lceil |e_1|/\sqrt{n} \rceil + \lceil |e_2|/\sqrt{n} \rceil \leq g. \quad (8)$$

Subsequently, the algorithm is recursively applied to e_1 and e_2, with $\lceil |e_1|/\sqrt{n} \rceil$ and $\lceil |e_2|/\sqrt{n} \rceil$ remaining partitions, respectively. Out of the eligible splits, we choose the most balanced one.

In the example of Figure 8.a there are $n = 9$ POIs, and 3 available buckets. The points are split into regions A, which contains $|A| = 3$ POIs, and BC, which contains $|BC| = 6$ POIs. BC is further split into B (where $|B| = 3$) and C (where $|C| = 3$). The resulting *kd*-tree has 2 levels. The root contains regions A, B, C and the leaf level contains 3 nodes with 3 POIs each, which are arranged in a PIR matrix M. Query processing follows the protocol of Figure 7. Since u is in region C, column 3 is retrieved; the NN is p_2.

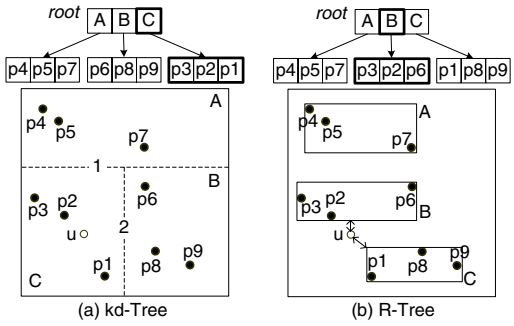

Figure 8: 2-D approximate NN

As another case study, consider the R-tree. Originally, each node would store between $f/2$ and f objects, where f is the node capacity; internal nodes contain minimum bounding rectangles (MBR) which enclose the objects of their children. We modify the R-tree construction algorithm such that there are 2 levels and the root contains no more than $\sqrt{n}$ MBRs. Let n' be the number of POIs in the current partition. The original algorithm checks all possible partitionings with $|e_1| + |e_2| = n'$ POIs, along the x and y-axis. It selects the best one (e.g., lowest total area, or total perimeter, etc) and continues recursively. We modify this algorithm to validate a split only if Equation 8 is satisfied. Figure 8.b shows an example where MBRs A, B, C contain 3 POIs each. The leaf nodes are arranged in a PIR matrix M and query processing follows the protocol of Figure 7. u is closer to MBR B, therefore column 2 is retrieved and the NN is p_2.

Both 2-D methods return the approximate NN by retrieving $\sqrt{n}$ POIs. The communication cost is $O(\sqrt{n})$. Therefore, in terms of cost, they are the same as the Hilbert-based method. The only difference is the approximation error, which depends on the characteristics of the dataset (e.g., density, skew). The case studies of the *kd*-tree and R-tree demonstrate a general method for accommodating any partitioning in our PIR framework. The choice of the appropriate partitioning for a specific dataset is outside the scope of this paper. Note that, all variations of ApproxNN can also return the approximate i^{th}-Nearest Neighbor, where $1 \leq i \leq \sqrt{n}$.

5. EXACT NEAREST NEIGHBORS

In this section we present a method, called *ExactNN*, which returns the POI that is the exact nearest neighbor of user u. In a preprocessing phase, ExactNN computes the Voronoi tessellation [7] of the set of POIs (see Figure 9). Every Voronoi cell contains one POI. By definition, the NN of any point within a Voronoi cell is the POI enclosed in that cell. ExactNN superimposes a regular $G \times G$ grid on top of the Voronoi diagram. Then, for every cell c of the grid, it determines all Voronoi cells that intersect it, and adds the corresponding POIs to c. Hence, cell c contains all potential NNs of every location inside it[2]. For example, Figure 9 depicts a 4×4 grid, where cell $A1$ contains $\{p_2\}$, cell $B2$ contains $\{p_2, p_3\}$, etc. During query processing, u learns the granularity of the grid; therefore he can calculate the cell that encloses his location (i.e., $D2$ in our example). Next, u

[2]ExactNN can be extended to support range queries, if each grid cell c stores the set of POIs it encloses

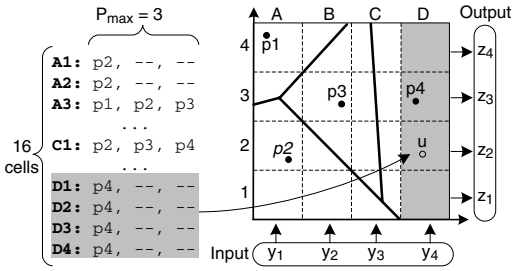

Figure 9: Exact nearest neighbor

issues a private request $PIR(D2)$; from the contents of $D2$ u finds his NN (i.e., p_4).

In contrast to ApproxNN methods, the objects of the PIR matrix M of ExactNN are not the POIs. Instead, each object in M corresponds to the contents of an entire grid cell c. For instance, our example contains 4 POIs (i.e., p_1, p_2, p_3, p_4), but M contains 16 objects, since there are 16 cells in the grid. In the previous section, n (i.e., the number of objects in M) was the same as the number of POIs. To avoid confusion, n still refers to the number of objects in M (i.e., $n = 16$ in the example) and we use $|POI|$ to denote the number of POIs.

All objects in M must have the same number of bits, otherwise the server may infer the requested cell based on the amount of bits transferred. Let P_{max} be the maximum number of POIs per grid cell. If a cell has fewer than P_{max} POIs, the server adds dummy POIs as placeholders. In our example, $P_{max} = 3$ because of cells $A3$ and $C1$. Therefore, all other cells are padded with dummy POIs. For instance, cell $A1$ becomes $\{p_2, -, -\}$. Recall from Table 1 that m denotes the number of bits of each object in M. Since there are P_{max} POIs in each grid cell, $m = |p_i| \cdot P_{max}$, where $|p_i|$ is the number of bits in the representation of each POI.

Since the number of objects in M is $n = G^2$, depending on the granularity of the grid, n may be larger or smaller than the number of POIs. P_{max} (hence m, too), also depends on G. Therefore the communication and computational cost of ExactNN depends on G. In Section 5.1 we discuss how to select an appropriate value for G.

Exact NN Protocol

User u:	Initiate query
Server:	Send grid granularity G
User u:	Let b be the column that includes u
	$y = [y_1 : y_{\sqrt{n}}]$, $y_b \in QNR$, and $\forall j \neq b, y_j \in QR$
	Send y
Server:	Send $z[1:m] = [z_1 : z_{\sqrt{n}}][1:m]$
User u:	Let a be the row that includes u
	Discard dummy POIs in z_a
	Calculate distance to real POIs in z_a
	Return the exact NN

Figure 10: Protocol for exact NN

The protocol for ExactNN is shown in Figure 10. It is similar to the ApproxNN protocol, with one difference: Let $\langle a, b\rangle$ be the cell that contains u, where a is the row and b the column. u issues a private request $PIR(\langle a, b\rangle)$. Recall that, in addition to $\langle a, b\rangle$, the byproduct of this request are the POIs of the entire column b. ApproxNN would utilize the extra POIs to improve the approximation of the result. On the other hand, the extra results are useless for ExactNN, since the exact NN is always in $\langle a, b\rangle$. A possible concern is that ExactNN reveals to the user $G \cdot P_{max}$ POIs, which may be more than those revealed by ApproxNN. In practice, however, this is not a problem because column b includes many duplicates. For example, cells $D1, D2, D3, D4$ in Figure 9 all contain the same POI p_4; therefore the request $PIR(D2)$ reveals only p_4 to the user. In Section 6.2 we discuss an optimization which reduces further the number of revealed POIs.

5.1 Grid Granularity

For a particular choice of grid granularity G, the PIR protocol overhead of ExactNN is $k \cdot G + k \cdot m \cdot G$ communication[3] (the first term corresponds to request y; the second to reply z), and $O(m \cdot G^2)$ server computation (recall that $m = |p_i| \cdot P_{max}$). By increasing G (i.e., finer grid), P_{max} may decrease or remain the same, depending on the data characteristics. Figure 11 shows the general form of the communication cost, as a function of G. Initially the cost decreases fast because P_{max} decreases, but later the cost increases again at finer granularity, as P_{max} reaches a lower bound (either 1, or the maximum of duplicate POIs). We could select the value of G that minimizes the communication cost, but there is a tradeoff, as the CPU cost increases quadratically to G. We could include the CPU cost in the graph and find the granularity that minimizes the total cost (expressed as response time). This would require the exact CPU speed and network bandwidth; the latter is problematic, since the bandwidth of each user differs. A good tradeoff is to select the granularity G_{opt} near the point where the rate of decrease of the communication cost slows down. That is the point where the slope of the tangent of the cost function becomes -1.

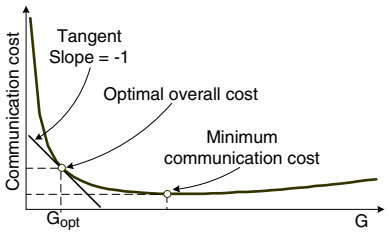

Figure 11: Finding the optimal grid granularity

In practice, since P_{max} is not known in advance, the graph of Figure 11 is generated as follows: First, we compute the Voronoi diagram of the dataset. Then, we select a set of values G_i using random sampling. For each of these values, we superimpose the resulting grid on the Voronoi diagram, and calculate P_{max} by counting the POIs in each cell. The communication cost is $C_i(G_i) = k \cdot G_i + k \cdot m \cdot G_i$. Finally, we apply curve fitting on the $\langle G_i, C_i(G_i)\rangle$ points to obtain the complete curve.

6. OPTIMIZATIONS

This section presents optimizations that are applicable to the previous methods. By employing these optimizations, the communication cost is reduced by as much as 90%, whereas the computational cost is reduced by up to 40% for a single CPU and more for multiple CPUs.

[3]Recall that k is the number of bits in the modulus

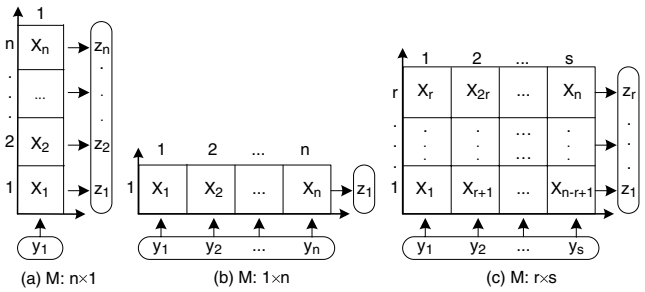

Figure 12: Rectangular PIR matrix M

6.1 Compression

The size of z (i.e., the server's answer) is $k \cdot m \cdot r$ bits, where r is the number of rows in the PIR matrix M. However, there is a lot of redundancy inside z. Consider the example of Figure 9. Cells $A4, B4, C4, D4$ have at least one dummy object each. The same holds for $A2, B2, C2, D2$. Assuming that the dummy object corresponds to bits $m_i \ldots m_j$, then $z_4[m_i : m_j]$ and $z_2[m_i : m_j]$ will be the same. Since each one of these results is k bits, the redundancy is significant. In our implementation we use standard compression techniques to compress the result. Our experiments showed that, in many cases, compression may save up to 90% of the communication cost.

6.2 Rectangular vs. Square PIR Matrix

In the previous sections the PIR matrix M is assumed to be square. However, M can have any rectangular shape [19] with r rows and s columns (see Figure 12). The shape of M does not affect the CPU cost, since the number of multiplications does not change. On the other hand, the communication cost becomes: $C(r, s) = k \cdot s + k \cdot m \cdot r$, where the first part is the size of the user's request $y_{1..s}$ and the second part is the size of the server's answer $z_{1..r}$. $C(r, s)$ is minimized for:

$$r = \left\lceil \sqrt{\frac{n}{m}} \right\rceil, \quad s = \left\lceil \frac{n}{r} \right\rceil \tag{9}$$

If each object has 1 bit (i.e., $m = 1$), $C(r, s)$ is minimized for $r = s = \sqrt{n}$ (i.e., square matrix). In our ExactNN method, on the other hand, $m \gg 1$; therefore, the communication cost is minimized for r smaller than s. Rectangular matrices have an additional benefit: they can reduce the number of POIs that the user learns. Consider the example of Figure 12.a, where $r = n$ and $s = 1$. The server returns $z_{1..n}$, therefore, the user learns n POIs. On the other hand, in Figure 12.b $r = 1$ and the server returns only 1 POI. By using rectangular M in the ExactNN algorithm, the user learns up to $r \cdot P_{max}$ POIs. This is much less than the $\sqrt{n} \cdot P_{max}$ POIs that a square matrix would reveal.

Rectangular M could also reduce the communication cost in the ApproxNN methods, since $m \gg 1$. However, there is a drawback: Recall that the ApproxNN methods organize POIs in an index, whose root node is always sent to the user. The size of the root is equal to the number of columns s. In the extreme case (i.e., for large enough m), Equation 9 results in $s \approx n$, therefore the root node reveals the entire database to the user. The minimum number of revealed POIs (i.e., $O(\sqrt{n})$) is achieved for square M. In our implementation we use a square matrix M for the ApproxNN methods.

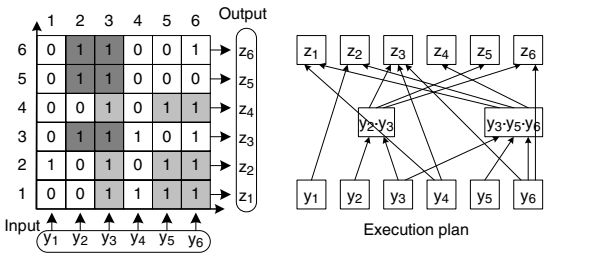

Figure 13: Pre-compiled optimized execution plan

6.3 Avoiding Redundant Multiplications

From Equation 6 (Section 3), it is clear that a PIR request requires $m \cdot n$ multiplications with $y_i \in y$. Each y_i is a k-bit number; to ensure that factorization is hard, k needs to be in the order of hundreds. Therefore, the CPU cost of the multiplications is high. Nevertheless, many multiplications are redundant, since they are repeated several times. In this section we propose an optimization technique, which employs data mining to avoid redundant multiplications. Although in this paper we only evaluate the effectiveness of the proposed optimization for the location privacy problem, our technique is general and can be used in other PIR applications.

By using the simplification of [24] (Section 3), in each row of the PIR matrix we only need to consider the '1' bits. For example, in Figure 13, the result for row 1 is: $z_1 = y_3 \cdot y_4 \cdot y_5 \cdot y_6$. Observe that the partial product $y_{356} = y_3 \cdot y_5 \cdot y_6$ appears in rows 1, 2 and 4. If y_{356} is computed once, it can be reused to compute $z_1 = y_{356} \cdot y_4$, $z_2 = y_{356} \cdot y_1$ and $z_4 = y_{356}$, thus saving many multiplications. The same idea applies to y_{23}, which appears in rows 3, 5 and 6.

Intuitively, the previous idea can be implemented as a "cache". When a new PIR request arrives, the server starts processing it and stores the partial results in the cache. If a partial product is repeated, the corresponding partial result is retrieved from the cache. Unfortunately, the number of possible partial products is 2^s, where s is the number of columns in M. s can be in the order of thousands, therefore the method is prohibitively expensive for on-line use.

Observe that, although the result depends on the input y, the set of multiplications depends only on the server's data and is the same for any PIR request. Therefore, similarly to pre-compiled query plans in databases, we generate in an off-line phase an optimized execution plan that avoids redundant multiplications. Then, during query processing, the server routes the input y through the operators of the plan, in order to compute fast the result z. The execution plan for our running example is shown in Figure 13.

In the off-line phase, we employ data mining techniques to identify redundant partial products. Following the data mining terminology, each item corresponds to one column of matrix M, whereas each transaction corresponds to a row of M. For example, row 1 in Figure 13 corresponds to transaction $T_1 = 001111$. A '1' bit means that the corresponding item belongs to the transaction. There are $r \cdot m$ transactions with s items each. An itemset corresponds to a partial product. In order to avoid many multiplications, we must identify frequent and long itemsets. We use the *Apriori* algorithm [2]. Initially, Apriori considers all itemsets with one item and prunes those that do not appear in at least f_{min} transactions. Then, it considers all possible combinations

with two of the remaining items and continues recursively with itemsets containing more items.

Accessing the execution plan incurs an overhead on query execution. Therefore, the frequency and length of the discovered itemsets must be large enough such that the savings from the multiplications are more than the overhead. The cut-off values for frequency and length can be estimated by measuring the actual multiplication time of the particular CPU. Moreover, by decreasing f_{min} the running time of Apriori increases. Therefore, f_{min} must be selected such that Apriori finishes within a reasonable time. Note that the identification of frequent itemsets is a costly operation, therefore it is not appropriate for databases with frequent updates. However, in many LBSs updates are infrequent (e.g., hospitals change rarely). Similar to data warehouses, our method is appropriate for batch periodic updates (e.g., once per night).

Let $IT = (it_1, it_2, \ldots)$ be the list of frequent itemsets sorted in descending order of itemset length. In the example of Figure 13, $IT = (001011, 011000)$ which corresponds to y_{356} and y_{23}. We use the following greedy algorithm to build the execution plan for row z_i: Let T_i be the transaction that corresponds to z_i. We traverse the list IT and select the first (i.e., longest) itemset it_j which appears in T_i. The rationale for this heuristic is that longer itemsets correspond to longer partial products, hence they are preferred for their higher potential in multiplication savings. We include it_j in the execution plan of T_i, remove from T_i all items in it_j (this step is necessary in order to ensure correctness) and repeat the process for the rest of itemsets in IT. The pseudocode is shown in Figure 14 (lines 3 and 5 use bitwise operations for performance reasons). The same process is repeated for all rows of M.

```
BuildExecutionPlan
Input: transaction T_i (from row i of M),
       list of frequent itemsets IT
1.  ExecPlan_i = ∅
2.  foreach itemset it_j ∈ IT
3.    if (¬T_i ∧ it_j = 0) /*it_j is part of T_i*/
4.      ExecPlan_i = ExecPlan_i ∪ {it_j}
5.      T_i = ¬it_j ∧ T_i
6.      if (T_i = 0) /*no more '1's in T_i*/
7.        break
8.  if (T_i ≠ 0)
9.    ExecPlan_i = ExecPlan_i ∪ {T_i}
10. output ExecPlan_i
```

Figure 14: Execution plan for one row

Figure 15 shows the architecture of the PIR optimizer. Once a query is received, the server checks for each row the associated execution plan $ExecPlan_i$: for each itemset $it \in ExecPlan_i$, the server checks whether the partial product of it has already been tabulated in table $PROD$; if so, it is used directly, otherwise, the server computes the product and stores it in $PROD$ to be used for subsequent rows. The overhead of this technique consists of the lookup in the $PROD$ table, which can be efficiently manipulated as a hash table, having as key the signature of it. The experiments show that, by using the optimized execution plan, the computation cost is reduced by up to 40%.

6.4 Parallelism

The PIR framework involves a large number of multiplications in a regular pattern. Consequently, the computations can be easily parallelized. The parallel computing infrastructure can vary from multicore CPU, to multi-CPU to computer cluster. Matrix M is partitioned horizontally in as many partitions as the available CPUs, and each CPU receives the corresponding partition in an off-line phase. During query processing, all CPUs receive the input vector y and calculate their part of the result. Communication is minimal (only the input and output) since each partition does not depend on the others. Therefore, parallel implementations achieve almost linear speedup. In our experiments we used up to 8 CPUs resulting to up to 7 times faster execution time.

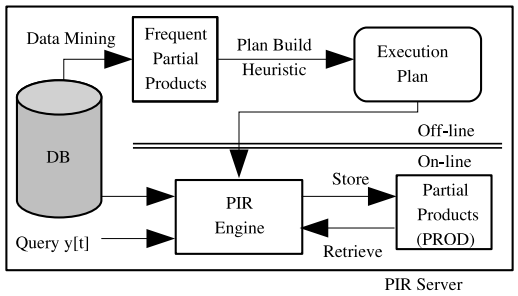

Figure 15: PIR Optimizer Architecture

7. EXPERIMENTAL EVALUATION

We developed a C++ prototype of the proposed PIR framework. We tested the methods using both synthetic (uniform and Gaussian) and real (Sequoia[4], 65K POIs in California) datasets. Our experimental testbed consisted of a Pentium 4 3.0GHz machine with 2GB of RAM, running Linux OS. We employed the *GMP*[5] library for operations with large integers (required by PIR), and the *zlib*[6] library for data compression. In our experiments, we measured the communication cost, as well as the computational cost at the server, which is the dominating factor for PIR. The CPU time includes the compression of the result before returning it to the client (which only accounts for a small fraction of the total CPU time). We also measured the computational cost at the client. We varied k (i.e., modulus bits) between 256 and 1280, and the number of POIs between $10,000$ and $100,000$. Each POI consists of its (x, y) coordinates (i.e., 64 bits).

7.1 1D and 2D Approximate NN

First we compare the approximate NN methods. 1D refers to the Hilbert variant, whereas 2D refers to the R-tree variant. Figure 16.a shows the server CPU time with varying k for the real Sequoia set. Recall that, for approximate methods, n is the number of POIs. The CPU time is very similar for both 1DApprox and 2DApprox, since in both cases it mainly depends on the data size. CPU time varies approximately as $k\sqrt{k}$, which is the average complexity of the multiplication algorithms implemented in GMP.

Figure 16.b, shows the communication cost, which is linear to k. Both methods incur similar cost, since the sets of elements (i.e., POIs) stored in the PIR matrix are the same in both cases. The slight difference is due to the distinct distribution of POIs along rows and columns. For $k = 768$, the communication cost is 1MB.

[4]http://www.rtreeportal.org
[5]http://gmplib.org/
[6]http://www.zlib.net

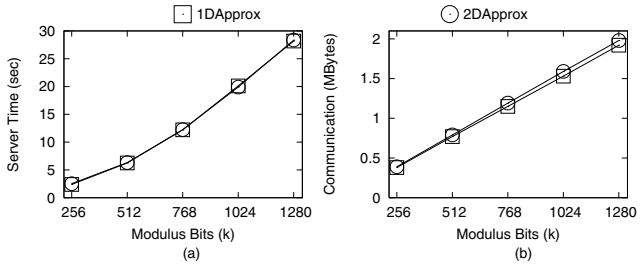

Figure 16: Variable k, Sequoia set (62K POI)

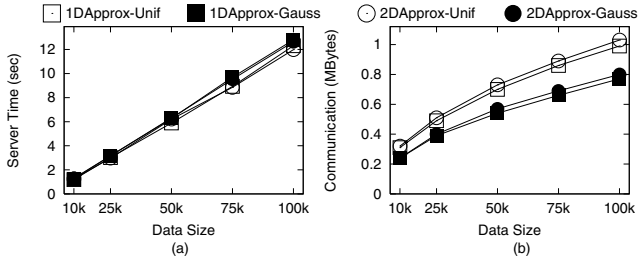

Figure 17: Variable data size, $k = 768$ bits

Figure 17.a shows the CPU time for varying data size (synthetic sets) and $k = 768$. The CPU time is linear to n, since the number of multiplications is proportional to the number of '1' bits in the data. The communication cost follows the expected theoretical dependency of $\sqrt{n}$, as shown in Figure 17.b. Compression is more effective with Gaussian data, because there are more POIs with nearby (possibly identical) coordinates, increasing redundancy.

Next, we investigate the approximation error of the proposed techniques. We generate 1000 queries originating at random locations that follow the POI distribution (this is a reasonable assumption, since the dataset is likely to correspond to an urban area, for instance). Given query point q, the returned result r and actual NN p, we express the approximate NN error as $err = (dist(q,r) - dist(q,p))/maxD$, where $maxD$ is the side of the (square) data space.

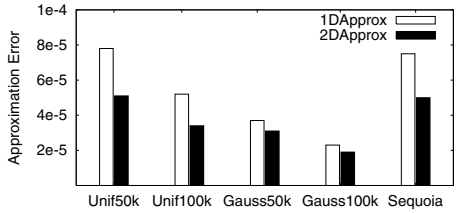

Figure 18: Approximation Error

Figure 18 shows the *average* error for 1DApprox and 2DApprox. The error is slightly larger for uniform data, as POIs are scattered in the entire dataspace. For Gaussian data, the clustering of POIs in the PIR matrix is more effective, leading to better accuracy. The error decreases when data density increases, and the average error is always below 0.01% of the dataspace size. Furthermore, the results revealed that despite their approximate nature, the methods return the exact NN for 94% of the queries, whereas the maximum *worst-case* error encountered was 1.1% of the dataspace size.

1DApprox and 2DApprox have similar CPU time and comparable communication cost, since they both follow the same 2-level tree approach. The choice between the two depends on the characteristics of the data and is outside the scope of this paper. In the rest of the experiments, we only consider the 1DApprox method.

	10K	25K	50K	75K	100K
Uniform	20x20	22x22	28x28	32x32	36x36
Gaussian	42x42	61x61	78x78	108x108	122x122
Sequoia	104x104				

Table 2: Grid Granularity for ExactNN

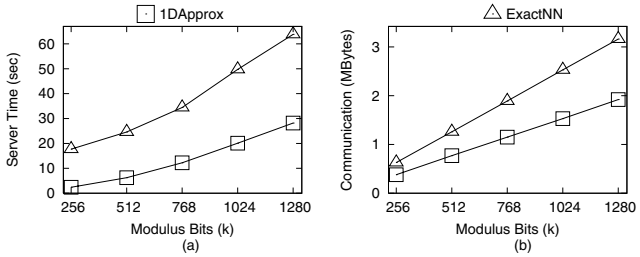

Figure 19: Variable k, Sequoia set (62K POI)

7.2 Exact Methods

We evaluate the performance of ExactNN in comparison with 1DApprox. The grid size of ExactNN was determined as described in Section 5.1. Table 2 shows the resulting grid size for each dataset. Note that, for ExactNN we use the rectangular PIR matrix optimization from Section 6.2. Figure 19.a depicts the CPU time versus k for the real dataset. The trend is similar to approximate methods, but the absolute values are higher for ExactNN, due to the larger size of the PIR matrix (recall that the m value for ExactNN may be considerably larger than that for 1DApprox). In Section 7.3 we evaluate methods that reduce the CPU time. Figure 19.b confirms that the communication cost is linear to k.

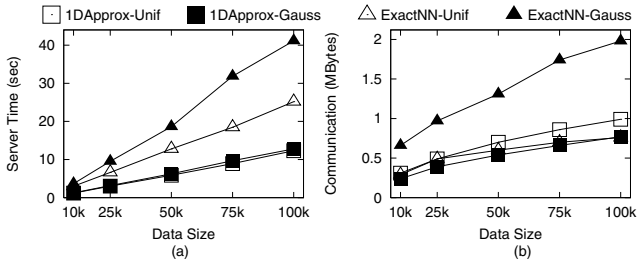

Figure 20: Variable data size, $k = 768$ bits

Figure 20.a shows the CPU time versus the data size. Recall that n for ExactNN depends on the grid granularity, and is *not* equal to the data size. For uniform data, the number of grid cells (i.e., n value) required to maintain a constant P_{max} grows proportionally with data size, therefore the CPU time increases linearly. On the other hand, for skewed data, in order to maintain a value of m which provides low communication cost, it may be necessary to use a finer grid, resulting in increased CPU time. However, the results show that the CPU time is almost linear to the number of POI, confirming that the heuristic for choosing the grid granularity is effective. The good choice of granularity is also reflected in the communication cost (Figure 20.b). Observe that, for Gaussian data, P_{max} (hence m) increases, and consequently the communication cost increases.

7.3 Execution Time Optimizations

In this experiment we evaluate our optimizer[7], which employs data mining (DM) to reduce the CPU cost of PIR at the server. We run the Apriori algorithm on the real dataset and retain all frequent itemsets with a support of at least 5%. Figure 21 shows the results: for small k values, the gain in execution time is less significant, because multiplications are relatively inexpensive. However, as k increases, the benefit of avoiding redundant multiplications becomes clear: the CPU time is reduced by up to 41% for 1DApprox, and 32% for ExactNN.

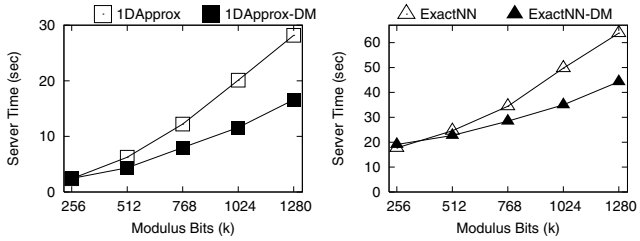

Figure 21: DM Optimization, Sequoia set

The PIR computations are suitable for parallel execution. We implemented a *Message Passing Interface (MPI)* version of the server, and tested it on a Linux cluster with Intel Xeon 2.8 GHz nodes. In Figure 22, we show the effect of parallel processing. We vary the number of CPUs from 1 to 8; note that, since each individual CPU is slower than the one used in the previous experiments, the 1-CPU time is slightly larger. The speed-up obtained is almost linear for 1DApprox, where we obtained improvements by a factor of 7.25 for 8 CPUs. For ExactNN, the speed-up is slightly lower, up to 6.1 for 8 CPUs, because the dummy objects correspond to a lot of '0' bits and result in load imbalance among the CPUs. We expect better performance with a more sophisticated load-balancing algorithm. For a typical value of $k = 768$ bits, 1DApprox finishes in 1sec, whereas ExactNN needs 6sec.

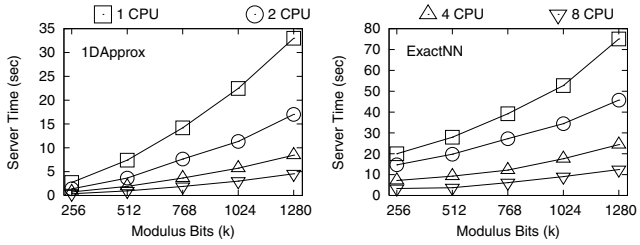

Figure 22: Parallel execution, Sequoia set

7.4 User CPU Time

The user is typically equipped with a slow PDA; therefore he cannot afford expensive computations. However, our experiments show that the CPU cost for the user is low. In Figure 23.a we use the real dataset and vary k. The user needs to generate random k-bit numbers and perform QR/QNR verifications of the k-bit replies. For typical $k = 768$, the CPU time does not exceed 0.6sec. In Figure 23.b we set $k = 768$ and vary the data size (we use the Gaussian dataset). When the data size increases, so does the number of columns in matrix M. Consequently, the size of the query vector y , as well as the size of the reply vector z, increases. The resulting CPU time is always lower than 0.4sec.

[7]For the execution time optimization experiments, we use square PIR matrices for both 1DApprox and ExactNN

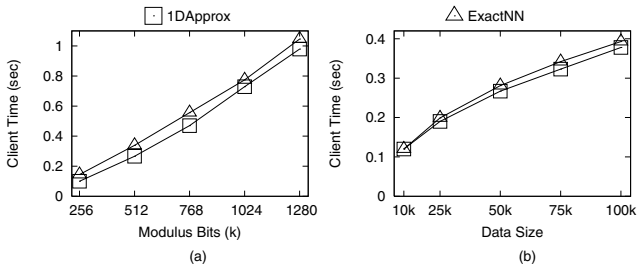

Figure 23: User CPU time

7.5 PIR vs. Anonymizer-based Methods

We compare our methods with *Hilbert Cloak (HC)* [17], which offers privacy guarantees for snapshot queries, and outperforms other cloaking-based location privacy techniques in terms of overhead, i.e. size of cloaking region (CR). Direct comparison is difficult, since the architectures are completely different and there are many unknowns (e.g., how many users subscribe in the anonymizer service, how often they update their location, how often they ask private queries, etc). Instead we study the number of POIs that the user learns from each query (recall from Section 3 that the user is charged by the number of retrieved POIs).

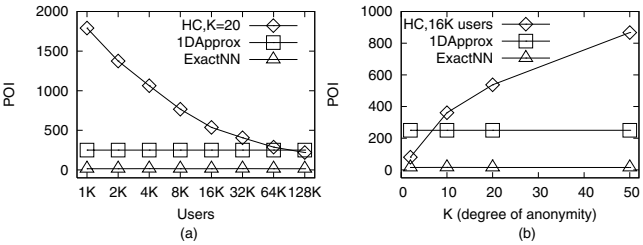

Figure 24: PIR vs. K-anonymity, Sequoia set

We consider the Sequoia dataset, and for HC we generate a number of subscribed users between 1K and 128K, at locations that follow the POI data distribution (as discussed in Section 7.1). Figure 24.a shows the number of disclosed POI for varying number of subscribed users, and a value of anonymity degree K of 20 (i.e., 5% probability of identifying the source). If the number of subscribed users is low, the size of the generated CR is large, and a huge number of POIs are included in the result. Only for a very large number of subscribers does the POI count become comparable with that of 1DApprox, which is roughly 250 for the Sequoia set. The number of disclosed POIs is even lower for ExactNN (i.e., 15 POIs in average), due to the rectangular PIR matrix. This result shows that, in order to maintain a reasonable degree of disclosed POIs (i.e., a compact CR), cloaking-based methods need to have a large number of subscribed users. This translates into a high cost of location updates (mobile users change location frequently), and also poses privacy concerns, since all users must be trustworthy. The disclosed POI number is constant for PIR methods, because no subscribed users are required.

In Figure 24.b we fix the number of subscribed users to 16,000 and vary K. HC discloses the same number of POI

as ExactNN for $K < 10$, which means that the identification probability of HC exceeds 10%. However, the identification probability of ExactNN is $\frac{1}{|U|} \ll \frac{1}{K}$, where U is the set of all possible users (see Section 3.2).

7.6 Discussion

The experimental results show that, although our PIR techniques are relatively expensive compared to usual query execution, the overhead is still reasonable. For the real dataset and a typical value of $k = 768$ bits, the communication cost for 1DApprox and ExactNN is roughly 1MB and 2MB, respectively. The corresponding CPU time at the server is 1sec and 6sec, respectively (by employing optimization and/or using multiple CPUs). The CPU time at the user is 1sec at most, and the number of disclosed POIs (hence the resulting financial cost of using the LBS), is low.

Existing cloaking-based approaches have many hidden efficiency issues, such as handling location updates, and managing a large number of user requests. In addition, existing methods have important drawbacks of qualitative nature: First, they lack privacy guarantees for continuous queries (i.e., correlation attack), and fail completely if some of the users are malicious. Second it may not be commercially feasible to gather the required large number of subscribers who will offer continuously their resources for a sporadic benefit. Third, there may be legal reasons which prohibit the anonymizer to gather locations of users.

8. CONCLUSIONS

In this paper, we employed the Private Information Retrieval theory to guarantee privacy in location-dependent queries. To the best of our knowledge, this is the first work to provide a practical PIR implementation with optimizations that achieve reasonable communication and CPU cost. Compared to previous work, our architecture is simpler, more secure (i.e., does not require an anonymizer or collaborating trustworthy users), and is the first one to protect against correlation attacks.

Currently, we are working on sophisticated heuristics to generate better optimized execution plans, in order to reduce further the CPU cost. In the future, we plan to investigate the extension of our framework to different types of queries, such as spatial joins.

9. REPEATABILITY ASSESSMENT RESULT

The results in Figures 16–21 and Figure 23 were verified by the SIGMOD repeatability committee. The committee has been unable to repeat the experiment described in Figure 22 due to the lack of appropriate hardware.

10. REFERENCES

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the k th-Ranked Element. In *Proc. of Int. Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 40–55, 2004.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of ACM SIGMOD*, pages 207–216, 1993.

[3] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar. Preserving user location privacy in mobile data management infrastructures. In *Int. Workshop on Privacy Enhancing Technologies*, pages 393–412, 2006.

[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[5] C.-Y. Chow and M. F. Mokbel. Enabling Private Continuous Queries for Revealed User Locations. In *Proc. of SSTD*, pages 258–275, 2007.

[6] C.-Y. Chow, M. F. Mokbel, and X. Liu. A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services. In *ACM International Symposium on Advances in Geographic Information Systems*, 2006.

[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[8] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proc. of ACM PODS*, pages 216–226, 1996.

[9] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright. Secure Multiparty Computation of Approximations. In *Int. Colloquium on Automata, Languages and Programming (ICALP)*, 2001.

[10] D. E. Flath. *Introduction to Number Theory*. John Wiley & Sons, 1988.

[11] B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *Proc. of ICDCS*, pages 620–629, 2005.

[12] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous Location-based Queries in Distributed Mobile Systems. In *Proc. of Int. Conference on World Wide Web (WWW)*, pages 371–380, 2007.

[13] O. Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[14] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proc. of USENIX MobiSys*, 2003.

[15] H. Hu and D. L. Lee. Range Nearest-Neighbor Query. *IEEE TKDE*, 18(1):78–91, 2006.

[16] P. Indyk and D. P. Woodruff. Polylogarithmic Private Approximations and Efficient Matching. In *Proc. of Theory of Cryptography Conference (TCC)*, pages 245–264, 2006.

[17] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing Location-Based Identity Inference in Anonymous Spatial Queries. *IEEE TKDE*, 19(12):1719–1733, 2007.

[18] A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *Proc. of SSTD*, pages 239–257, 2007.

[19] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: Single database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[20] M. F. Mokbel, C. Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *Proc. of VLDB*, 2006.

[21] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE TKDE*, 13(1):124–141, 2001.

[22] P. Samarati. Protecting Respondents' Identities in Microdata Release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[23] M. Shaneck, Y. Kim, and V. Kum. Privacy Preserving Nearest Neighbor Search. In *Int. Workshop on Privacy Aspects of Data Mining (PADM)*, 2006.

[24] R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2007.

[25] J. Vaidya and C. Clifton. Privacy-Preserving Top-K Queries. In *Proc. of ICDE*, pages 545–546, 2005.