Location privacy: Going beyond K-anonymity, cloaking and anonymizers

3 authors, including:

Ali Khoshgozaran
University of Southern Calif…
16 PUBLICATIONS   **581** CITATIONS

Cyrus Shahabi
University of Southern Calif…
374 PUBLICATIONS   **6,227** CITATIONS

Location privacy: going beyond K-anonymity, cloaking and anonymizers

**Ali Khoshgozaran · Cyrus Shahabi ·
Houtan Shirani-Mehr**

Abstract   With many location-based services, it is implicitly assumed that the location server receives actual users locations to respond to their spatial queries. Consequently, information *customized* to their locations, such as nearest points of interest can be provided. However, there is a major privacy concern over sharing such sensitive information with potentially malicious servers, jeopardizing users' private information. The anonymity- and cloaking-based approaches proposed to address this problem cannot provide stringent privacy guarantees without incurring costly computation and communication overhead. Furthermore, they require a trusted intermediate anonymizer to protect user locations during query processing. This paper proposes a fundamental approach based on *private information retrieval* to process range and K-nearest neighbor queries, the prevalent queries used in many location-based services, with stronger privacy guarantees compared to those of the cloaking and anonymity approaches. We performed extensive experiments on both real-world and synthetic datasets to confirm the effectiveness of our approaches.

Keywords   Location privacy · Spatial databases · Location-based services ·
Private information retrieval

1 Introduction

The explosive growth of affordable GPS-enabled cell phones has resulted in a variety of innovative applications based on user locations. Almost all of these location-based services

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

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

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

(LBS) use location servers that are somehow *aware* of their users locations in order to provide customized services. However, recent concerns over how such potentially untrusted *location servers* can jeopardize a user's private information resulted in a newly coined term of *location privacy*. Several breaching of subscribers privacy by misusing their location information have been reported [8,30], and many researchers and organizations have raised the need to explore the privacy threats associated with location-based services [39,40] and go as far as warning about the creation of "a *ubiquitous surveillance* system" if user privacy in ubiquitous mobile systems is not carefully addressed [1].

In addition to revealing an individual's location, LBS queries such as "find the nearest cancer treatment center" may disclose other sensitive information about individuals including health condition, lifestyle habits, political and religious affiliations, or may result in unsolicited advertisements (spam). It is important to note that hiding user identities alone without hiding user locations would not address these privacy issues. An attacker or a (potentially malicious) location server can *infer* the identity of the query source from its (subsequent) location information [15]. For example, a user's location information can be associated with a certain residence or office location and easily lead to determine the user's identity. Several other types of surprisingly private information can also be revealed by just observing anonymous user movements and cell phone usage patterns over time (e.g., www.bluetoothtracking. org).

There is a large class of approaches to enable location privacy that are based on the idea of hiding user locations in a larger (and thus harder to track) region (*cloaking*) or among a set of other users (*anonymity*). However, several recent studies [14,17,25,26] have shown that such approaches suffer from many drawbacks such as an insufficient guarantee of privacy, vulnerability to correlation attacks, the need for a sophisticated location anonymizer and a huge performance hit for privacy paranoid users.

We use *Private Information Retrieval* (PIR) techniques to provide a fundamentally more generalized and powerful way of blinding the untrusted location server by converting spatial query processing into several private database retrievals from the location server. A PIR protocol allows a client to secretly request a record stored at an untrusted server without revealing the retrieved record to the server. Therefore, instead of only blurring user queries (as in anonymity and cloaking approaches), we use PIR to protect the queried *content*. This way, no information is leaked to adversaries by examining the records requested from the server. However, anonymity and cloaking approaches, by design, cannot avoid this information leakage to achieve perfect secrecy, regardless of the underlying information-hiding technique used.

The use of PIR techniques as a fundamentally novel approach to protect user privacy in location-based services was first proposed in [26] and [14]. However, as we show in this paper, the approach proposed in [14] has three important restrictions compared to our approach: (a) it is limited to blind evaluation of *first* nearest neighbor queries (b) it cannot avoid a linear scan of the entire database for processing each query (c) the communication complexity of each query is also very high (roughly $\sqrt{n}$ where n represents the size of the dataset). We extend our initial work in [26] and propose several algorithms to efficiently process range and K-nearest neighbor (KNN) queries. Therefore, our framework supports a more general class of queries fundamental to many location-based services while satisfying similar stringent privacy guarantees provided by [14] against most powerful adversaries. We empirically compare our proposed approaches with [14] for 1NN queries and elaborate why they incur significantly less communication and computation complexity. The above benefits come with the cost of utilizing a Secure Coprocessor to execute the PIR scheme. Although we utilize secure coprocessors and a specific PIR scheme, the PIR module can be treated as a

black box throughout the query resolution process and thus any other practical PIR approach (that is either proposed or will emerge) can replace our current PIR scheme.

While PIR can be used to privately generate a query result set, avoiding a linear private scan of the entire database is challenging. This is due to the fact that the server owning the objects information cannot be trusted to perform the query processing and choose what to be queried. Alternatively, moving this knowledge to the users will require the query processing to happen at the client side which results in high communication cost for transferring queried information [14]. Utilizing PIR, we employ three alternative index structures that greatly reduce the amount of information that is privately queried from the untrusted server. We have both analytically studied the performance of these index structures and performed extensive sets of experiments on real-world and synthetic datasets to empirically verify our analytical results. We show that although PIR constitutes around 90% of the overall query processing time, the total query response time is still in the order of milliseconds with proper system parameters.

A preliminary version of this paper appeared in [26]. This paper subsumes [26] by presenting two new private index structures as well as three KNN query processing algorithms that use these index structures. Finally, we conducted sets of new experiments with three new datasets and experimentally compared our techniques with several related studies.

The remainder of this paper is organized as follows. Section 2 provides a background on location privacy preliminaries and PIR. In Sect. 3, we propose our private index structures and several algorithms to privately evaluate range and KNN queries. In Sect. 4, we discuss how to derive optimal system parameters for our framework and Sect. 5 shows how the aforementioned algorithms incorporate a practical PIR scheme to enable location privacy. Section 6 includes our experimental results. In Sect. 7, we review the current research in location privacy and finally Sect. 8 discusses the conclusions and our future plans.

2 Background

In this section, we briefly provide some background information related to our proposed privacy requirements which can be used to evaluate any location privacy framework and review PIR as the foundation of our framework.

The obvious objective of any privacy-aware LBS is to protect a user's private information from potentially malicious servers while responding to his queries. In order to achieve location privacy, user location and identity information, as well as the identity of query results should be kept secret both on the server and during query evaluation. We employ PIR to achieve such strong measures of privacy by placing trust on a secure coprocessor residing at the server side which is in charge of initiating PIR requests to the server and privately evaluating user queries (Fig. 1). To this end, we define the privacy metrics, the adversary and the information leak model and use them throughout the paper to evaluate the privacy of our proposed approaches.

2.1 Threat model and privacy metrics

We consider a model in which users query a central untrusted location server S. Users subscribe to S's services and form range or KNN queries using their handheld devices. While users trust their client devices to run legitimate software, they do not trust any other entity including S. Users might collude with S against other users and thus from each user's point of view, all other users and the server can be adversarial. As part of our threat model, we

Fig. 1 Privacy aware LBS

assume that the server's database is publicly accessible and available and thus the adversary can perform a known plaintext attack. We also assume the strongest adversary, which consists of a malicious user who subscribes to the system as a normal user colluding with S.

Given a database of objects DB $= (o_1, o_2, \ldots, o_n)$ in a 2-D space hosted by S and a set of users $U = (u_1, u_2, \ldots, u_m)$, we try to satisfy the (slightly modified variants of) privacy requirements proposed in [25] to ensure that evaluating a spatial query does not reveal any sensitive location information to the potentially untrusted server. Each o_i represents an object (such as a restaurant or a gas station) by the triplet $\langle longitude, latitude, id \rangle$.

Definition 1 *u-anonymity*: While resolving a query, the user issuing the query should be indistinguishable among the entire set of users. In other words, for each query q, $P_q(u_j) = \frac{1}{m}$ where $P_q(u_j)$ is the probability that query q is issued by a user u_j where $j \in \{1 \ldots m\}$ and m is the number of users.

Note that this definition ensures the server does not know which user issued the query q; however, we also need to ensure that the server does not know from which point the query q is issued. This requirement is captured by Definition 2.

Definition 2 *a-anonymity*: While resolving a query, the location of the query point should not be revealed. In other words, for each query q, $P'_q(l_i) = \frac{1}{area(A)}$, where A is the entire region covering all the objects in DB and $P'_q(l_i)$ is the probability that the query q was issued by a user located at a point l_i inside A.

Definition 3 *Blind evaluation of spatial queries*: We can now define *blind evaluation of spatial queries* as satisfying *u-anonymity* and *a-anonymity* constraints while resolving spatial queries. A location server is termed *privacy aware* if it is capable of blindly evaluating spatial queries. It is clear now that location privacy is achieved if all spatial queries issued by users are evaluated blindly.

Note that Definitions 1 and 2 impose much stronger privacy requirements than the commonly used K-anonymity metric [12,16,24,28], in which a user is indistinguishable among a small set of K other users or his location is blurred in a small cloaked region R. The above definitions of location privacy are in fact identical to an extreme case of setting $R = A$ for spatial cloaking (i.e., extending the possible user location to the *entire* region) or $K = m$ for K-anonymity (i.e., making a user indistinguishable among *all* users). Satisfying these stringent privacy requirements is the key distinguishing factor between our approach and the privacy guarantees of anonymity-based approaches.

Definition 4 *Information leak*: It is important to be able to measure how much private information is revealed by performing the necessary steps in responding to a spatial query. Similar to [3], we use entropy to measure how much information is *leaked* by following a privacy preserving protocol. A set of queries $Q = (q_1, q_2, \ldots, q_k)$ is privately evaluated if and only if the joint entropy of the variables $q_1, q_2, \ldots, q_k$ is maximal. This definition of information leak captures the "absence of information about a set of queries".

Using the definitions above, we can now more formally discuss our motivation behind using private information retrieval to achieve location privacy. As we discuss in Sect. 7, a drawback inherent in cloaking- and anonymity-based approaches to location privacy is the information leak where an adversary with strong prior knowledge is able to infer sensitive user location information based on the information queried from the server.

As discussed above, our motivation is to achieve a significantly more stringent user location privacy through PIR and converting spatial query processing into several private database retrievals. Therefore, one of the key challenges behind such a framework is devising spatial algorithms that enable query evaluation using a privacy-aware server that only responds to private object information retrievals and does not possess any location information. In the following section, we provide an overview of several PIR schemes and the one we utilize in our framework.

2.2 Private information retrieval

Suppose Bob owns a database DB of n objects and Alice is interested in DB[i]. Although Bob might know the entire content of DB, Alice is not willing to disclose i to Bob. A Private Information Retrieval (PIR) protocol allows Alice to *privately* retrieve DB[i] from Bob ensuring he does not learn the value of i.

The PIR problem was first proposed by Chor et al. [9] in an information-theoretical setting, which also proves that any theoretical PIR scheme has a lower communication bound equal to the database size although it provides perfect secrecy against an adversary with unbounded computational power. In order to mitigate the communication cost, computational PIR schemes consider a computationally bounded adversary where the security of the approaches relies on the intractability of a computationally complex mathematical problem, such as Quadratic Residuosity Assumption [27]. However, similar to information-theoretical PIR, this class of approaches cannot avoid a linear scan of all database items per query. To obtain perfect privacy while avoiding the high cost of the approaches discussed above, a new class of Hardware-based PIR approaches has recently emerged which places the trust on a tamper-resistant hardware device. These techniques benefit from highly efficient computations at the cost of relying on a hardware device to provide privacy [3,4,18,34]. Placing a trusted module very close to the untrusted host allows these techniques to achieve optimal computation and communication cost compared to the computational and theoretical PIR approaches. Therefore, we employ hardware-based PIR techniques as the building block for our privacy-aware location server to achieve acceptable communication and computation complexity. However, it is important to note that we treat the PIR module as a black box throughout the query resolution process and thus any other practical PIR scheme can be incorporated into our current framework. For now we assume that the operation $read(DB_\pi, i)$ privately retrieves the ith element of the encrypted database DB_π stored at the untrusted server using the hardware-based PIR scheme. We use t_{read} to denote the time it takes to perform a private read from the database. In Sect. 5, we detail how a secure coprocessor is

used to enable hardware-based PIR and provide more details about DB_π as well as how the complexity of the *read* operation performed on such a database.

3 Privacy-aware query processing

So far we have enabled private retrieval from an untrusted server. However, we have not focused on how spatial queries can be evaluated privately. Section 2.2 enables replacing a normal database in a conventional query processing with its privacy-aware variant. However, the query processing needs to be able to utilize this new privacy-aware database as well. Note that what prevents us from using encrypted databases is the impossibility of blindly evaluating a sophisticated spatial query on an encrypted database without a linear scan of all encrypted items.

In this section, we employ private index structures that enable blind evaluation of spatial queries efficiently and privately. Using these index structures, we devise a sweeping algorithm to process range queries and three algorithms; *Progressive*, *Hierarchical* and *Hilbert-based* (or Hilbert for short) to privately evaluate KNN queries. We assume mobile users subscribe to the untrusted location server to query points of interest (POI) data such as restaurants and hospitals.

The key idea behind our approach is to use PIR to privately query the index structures (Sect. 3.1) stored at the untrusted server to perform spatial queries. The algorithms discussed in this section employ the *read*() function (Sect. 2.2) to privately retrieve relevant records from the server's database. The choice of where these algorithms are executed depends on the underlying PIR protocol employed. As we elaborate in Sect. 5, we place trust on a secure coprocessor (SC) residing at the server side to execute the range and KNN algorithms. We defer the discussion of users, server and SC's role in our hardware-based implementation of PIR to Sect. 5. Note that a PIR protocol guarantees that the server cannot gain any information about what records are actually being retrieved for query processing. In the following section, we show how we employ private spatial index structures to avoid a linear scan of the entire database records that are hosted at the untrusted server.

3.1 Index structures

In this section, we present three index structures that allow blind evaluation of range and KNN queries. Each index structure is constructed, encrypted and stored at the untrusted host during a preprocessing step. These index structures allow us to efficiently scan only a subset of the records stored in DB_π at the untrusted server while processing spatial queries. Later in Sect. 3.3, we show how each KNN query reduces to a range query which itself reduces to several private reads from the database.

The range and three KNN algorithms discussed below utilize a regular grid structure to construct the indexing used by the query processing module. The key reason behind using a grid structure in our framework is while being efficient, grids simplify the query processing. While other spatial indexes such as r-trees and kd-trees provide efficient spatial query processing, they require the client to incrementally retrieve the underlying tree structure to further guide the search. This approach is very costly when using PIR, as aside from the data (i.e., object information), tree navigation should also be performed privately using PIR. As we show in Sect. 6.7, PIR significantly dominates the cost of query processing. Knowing the grid granularity, we can quickly convert a spatial query into private retrieval of certain records without requiring to privately query a tree-based representation of the objects first.
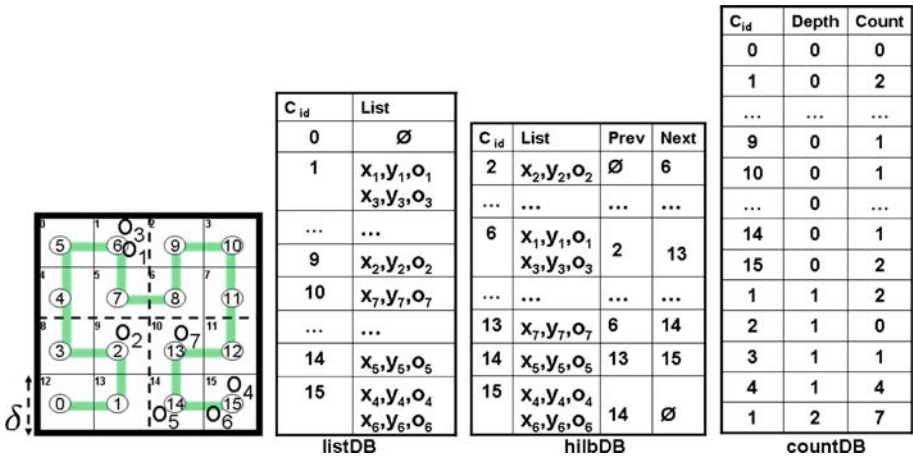
listDB

C_{id}	List
0	∅
1	x_1,y_1,o_1 x_3,y_3,o_3
...	...
9	x_2,y_2,o_2
10	x_7,y_7,o_7
...	...
14	x_5,y_5,o_5
15	x_4,y_4,o_4 x_6,y_6,o_6

hilbDB

C_{id}	List	Prev	Next
2	x_2,y_2,o_2	∅	6
...	...	...	...
6	x_1,y_1,o_1 x_3,y_3,o_3	2	13
...	...	...	...
13	x_7,y_7,o_7	6	14
14	x_5,y_5,o_5	13	15
15	x_4,y_4,o_4 x_6,y_6,o_6	14	∅

countDB

C_{id}	Depth	Count
0	0	0
1	0	2
...	...	...
9	0	1
10	0	1
...	0	...
14	0	1
15	0	2
1	1	2
2	1	0
3	1	1
4	1	4
1	2	7

Fig. 2 The private index structures

Several studies have shown the significant efficiency of using the grid structure for evaluating range, KNN and other types of spatial queries [23,41,43].

The spatial characteristics of indexing objects with regular grids make it challenging to readily use PIR to query grid cells. Later in Sect. 4, we detail how we modify our original private indexes developed in this section for performance and privacy reasons. More specifically, Sect. 4.1 discusses how to derive the optimum grid granularity according to the costs of different PIR and non-PIR operations and in Sect. 4.2, we propose a secure padding scheme to prevent a potential information leakage while using PIR to query grid cells.

Without loss of generality, we assume the entire area enclosing all objects is represented by a unit square. We uniformly partition the unit square into an $M \times M$ grid where each grid cell has side length δ $(0 < \delta \leq 1)$ such that $M = \frac{1}{\delta}$ is a power of 2 (see Fig. 2). Each cell is identified by its sequential cell ID (c_{id}) and may contain several objects each being represented by the triplet $\langle x_i, y_i, obj_{id} \rangle$. These cells are then used to construct *listDB*, *countDB* and *hilbDB* index structures.

The *listDB* index stores the objects and their location information for each cell. The *listDB* schema represents a flat grid and looks like $\langle c_{id}, list \rangle$ where $list$ is a sequence of triplets representing objects falling in each grid cell. Note that *listDB* does not store a hierarchical representation of aggregate information regarding the object distributions (e.g., cell density). Such hierarchical representation is beneficial in pruning a large portion of the search space during query processing. The next index structure is introduced to capture such aggregate cell information.

The *countDB* index, maintains a multi-level hierarchical index storing the number of objects in each cell at each depth. A cell at depth i is constructed by merging four adjacent cells of depth $i - 1$ and adding up their object counts. The intuition behind *countDB* is to keep count of the number of objects in each region (note that multi-level storage of all object coordinates for all depths in *listDB* causes redundancy and incurs a significant space overhead). The schema of *countDB* looks like $\langle c_{id}, depth, count \rangle$ where each tuple represents the number of objects falling in cell c_{id} at each depth. Since this schema is not consistent with our definition of PIR in which the key for any retrieval is only a single number (i.e., the ith element), we design a mapping to convert *countDB* into a plain key-value relation. Observe that starting from an $M \times M$ grid, at each depth d, the grid has $\frac{M}{2^d} \times \frac{M}{2^d}$ cells where

$d \leq \log_2 M$. Therefore, we devise the function *convert* (Eq. 1) to rewrite *countDB*'s schema as $\langle i, \text{count} \rangle$ where i is a unique cell identifier.

$$i = \begin{cases} c_{id} & d = 0 \\ \text{convert}(c_{id}, d) = \sum_{i=0}^{d-1} \left(\frac{M}{2^i} \times \frac{M}{2^i} \right) + c_{id} & d \geq 1 \end{cases} \quad (1)$$

Given a skewed dataset, a KNN algorithm utilizing the grid structure will experience a performance degradation caused by numerous empty cells in the grid. Increasing δ does not solve this problem as it results in coarse-grained cells containing many objects that have to be queried/processed and even a linear decrease in δ incurs at least a quadratic increase in the number of empty cells. The *hilbDB* index uses Hilbert space filling curves to avoid these shortcomings of processing KNN queries using regular grids (in particular for skewed datasets). The main intuition behind using Hilbert curves is to use their locality preserving properties to efficiently approximate the nearest objects to a query point by only indexing and querying the non-empty cells. As we show in Sect. 6, this property significantly reduces the query response time for skewed datasets.

We define H_2^N ($N \geq 1$), the Nth order Hilbert curve in a 2-dimensional space, as a linear ordering which maps an integer set $[0, 2^{2N} - 1]$ into a 2-dimensional integer space $[0, 2^N - 1]^2$ defined as $H = \nu(P)$ for $H \in [0, 2^{2N} - 1]$, where P is the coordinate of each point. The output of this function is denoted by *H-value*.

To create the *hilbDB* index, an H_2^N Hilbert curve is constructed traversing the entire space. After visiting each cell C, its $c_{id} = \nu(C)$ is computed using the center of C. We use an efficient bitwise interleaving algorithm from [11] to compute the H-values (the cost of performing this operation is $O(n)$ where n is the number of bits required to represent a Hilbert value). Next, similar to the *listDB* index, the c_{id} values are used to store object information for each cell. Finally, in order to guide the next retrieval, each record also keeps the index of its non-empty c_{id} neighbors in *hilbDB*, stored in the *Prev* and *Next* columns, respectively. These two values allow us to find out which cell to query next from *hilbDB* hosted at the server.

Figure 2 illustrates the original object space and the above three index structures. The circled numbers denote each cell's c_{id} constructed by H_2^2 for the *hilbDB* index. For clarity, we have kept the original 3-column format for *countDB* and also we have not shown that all records are in fact stored in an encrypted format.

3.2 Private range queries

Using the *listDB* index, processing range queries is straightforward. We use a sweeping algorithm to privately query the server for all cells which overlap with the specified range. A range query R is defined as a rectangle[1] of size $l \times w$ ($0 < l, w \leq 1$). Therefore, to answer each range query, using *listDB* we must first find the set of cells R' that encloses R. R' forms a $L \times W$ rectangular grid where $L \leq \lceil \frac{l}{\delta} \rceil + 1$ and $W \leq \lceil \frac{w}{\delta} \rceil + 1$ (see Fig. 3a). The function $read(listDB, c_{id})$ privately queries $listDB$ and performs the necessary processing to return a list of all objects enclosed in a c_{id}. We use a sweeping algorithm to query the cells in R' privately. Algorithm 1 provides the pseudocode for evaluating range queries.

---

[1] Without loss of generality, we assume the range query has a rectangular shape, circular regions can be queried by their enclosing rectangles and filtering the false positives at the client side.

Fig. 3 The range query R (**a**) computing the safe region R' for the KNN query (**b**)

(a)　　　　(b)
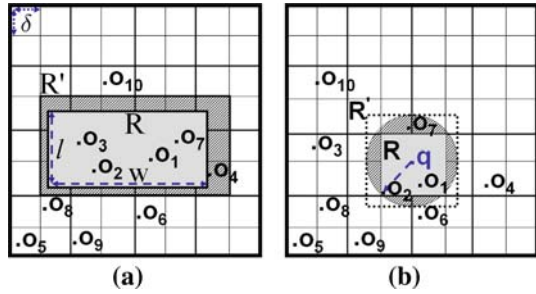
Algorithm 1 Range(R)

Require: $P_{ll}=R$'s lower left point, $P_{ur}=R$'s upper right point;

　　$S \leftarrow \emptyset$;

2: **for** $(col = \lceil \frac{P_{ll}.x}{\delta} \rceil; col \leq \lceil \frac{P_{ur}.x}{\delta} \rceil; col++)$ **do**

　　　for $(row = \lceil \frac{P_{ll}.y}{\delta} \rceil; row \leq \lceil \frac{P_{ur}.y}{\delta} \rceil; row++)$ **do**

4:　　　$c_{id} = (\frac{row-1}{\delta}) + col$;

　　　　$L = read(listDB, c_{id})$;

6:　　　**for all** $o_i \in L$ **do**

　　　　　if $((P_{ll}.x \leq o_i.x \leq P_{ur}.x)$ and $(P_{ll}.y \leq o_i.y \leq P_{ur}.y))$ **then**

8:　　　　　$S = S \cup \{o_i\}$;

　　　　　end if

10:　　　**end for**

　　　end for

12: **end for**

　　return S;

The time complexity of Algorithm 1 can be written as follows.

$$t_{\text{range}} = \left(\left\lceil \frac{l}{\delta} \right\rceil + 1 \right) \times \left(\left\lceil \frac{w}{\delta} \right\rceil + 1 \right) \times t_{\text{read}} = O\left(\frac{l \times w \times t_{\text{read}}}{\delta^2} \right) \qquad (2)$$

Since $R \neq R'$, for the range R of size $l \times w$, Algorithm 1 queries $L \times W$ cells which is linear with respect to $area(R)$. For a uniform distribution of n objects, each cell on average contains $n \times \delta^2$ items. Therefore, the total number of items queried is $O(\alpha \times n)$ for $\alpha = L \times W \times \delta^2$ which is also linear with respect to n.

3.3 Private KNN queries

The main challenge in evaluating KNN queries rises from the fact that the distribution of points can affect the size of the region R that contains the result set (and hence the cells that should be retrieved). In other words, no region is guaranteed to contain the K nearest objects to a query point (except in a uniform distribution) which implies that R has to be progressively computed based on object distributions. Therefore, it is important to minimize the total number of cells that should be privately queried. In Sects. 3.3.1 through 3.3.3, we examine three variants of evaluating KNN queries and discuss how each index structure allow us to query only a small subset of the entire object space. Note that due to the strong similarity of these algorithms with their *first nearest neighbor* counterparts, we directly consider the more general case of KNN.

The following general approach is utilized by each of our KNN algorithms: (a) create a region R and set it to the cell containing the query point q (b) expand R until it encloses at least K objects (c) compute the *safe region* R' as the region guaranteed to enclose the result set and (d) find the actual K nearest objects in R' using $range(R')$ defined above.

The main difference among the three algorithms is related to how they perform the step (b) mentioned above. Regardless of the approach, R is not guaranteed to contain the actual K nearest neighbors of q due to approximating the circular region around q with the rectangular region R (see Fig. 3b where $O_7 \in 2NN(q)$ but $O_7 \notin R$ and $O_2 \in R$ although $O_2 \notin 2NN(q)$). Therefore, R has to be expanded to a *safe region* R' which is the rectangle enclosing the circular region that contains the actual K nearest objects to the query point. As shown, if relative location of q and its furthest neighbor in R is known, a safe region can be constructed. It is easy to verify that R' is a square with sides $2 \times \lceil ||c_q - far_q(K)|| \rceil$ where c_q is the cell containing q and $far_q(K)$ is the cell containing q's Kth nearest object in R and $||.||$ is the Euclidean norm [43]. Once the safe region is computed, the objects located in R' are retrieved and added to the result set. We slightly modify the range algorithm to avoid querying cells previously checked during the computation of R. We now elaborate on how different expansion strategies for step (b) mentioned above generate different results and discuss the pros and cons of each strategy.

3.3.1 Progressive expansion

With this approach, if K objects are not found in the cell containing the query point, we expand the region R in a concentric pattern until it encloses K objects. The most important advantage of this method is its simple and conservative expansion strategy. This property guarantees that the progressive expansion minimizes R. However, the very same conservative strategy might also become its drawback. This is because for non-uniform datasets, it takes more time until the algorithm reaches a valid R. Algorithm 2 details the progressive expansion strategy. Once the querying cell is identified, objects within that cell are privately retrieved and added to the result set (lines 2–4). Next, the region starting from the query cell is expanded progressively (the *Expand* function) and objects in the examined cells are added to S until K objects are found (lines 5–10). Next, the safe region R' is first computed then queried and the final result set is computed (lines 11–12). Note that Algorithm 2 only uses the *listDB* index to evaluate a KNN query. The time complexity of this algorithm can be written as follows.

$$t_{progressive} = O\left(\frac{K}{n\delta^2} \times t_{read} + t_{range}\right) \tag{3}$$

The first term in Eq. 3 corresponds to the average number of private cell retrievals to find the first K objects and the second term denotes the time it takes to query the cells added by the safe region and construct the actual result set. Obviously, in contrary to t_{range}, the time complexity of processing KNN queries varies with n (i.e., the number of objects in the database). Also note that due to the conservative and symmetric expansion, $|R' - R|$ is relatively small compared to $|R|$.

3.3.2 Hierarchical expansion

The hierarchical approach takes a completely different strategy to construct the region R. Given the query point q, at each step if K items are still not found, the algorithm moves one level higher in the cell hierarchy and picks q's parent, grand parent, etc. until R encloses K

Algorithm 2 KNN-Progressive(K, q)

Require: K, q;
   $S \leftarrow \emptyset; cnt \leftarrow 0$;
   $cell.x = \lceil \frac{q.x}{\delta} \rceil; cell.y = \lceil \frac{q.y}{\delta} \rceil$;
3: $c_{id} = (\frac{cell.y-1}{\delta}) + cell.x$;
   $S = read(listDB, c_{id})$;
   let $region \leftarrow cell_{c_{id}}$;
6: $cnt = |S|$;
   while $(cnt < K)$ **do**
     $region = Expand(region)$;
9:   $S = S \cup read(listDB, region.cellids_{new})$;
     $cnt = |S|$;
   end while
12: $R' = safeRegion(S)$;
   return $Range(R')$;

---

objects. The advantage of moving from a cell to its parent lies in the fact that constructing R reduces to very few reads of the *countDB* index instead of progressively counting the number of objects in each regular cell privately. Furthermore, this approach trivially overcomes the limitation of the progressive expansion and very quickly converges; however, it suffers from another drawback. The exponential increase in the size of R results in a relatively large R' which in turn increases the overhead of privately retrieving numerous cells in R'. The time complexity of the hierarchical algorithm is given in Eq. 4. Note that due to its expansion strategy, the t_{range} term here is significantly higher than t_{range} from Algorithm 2. Furthermore, in contrast to Algorithm 2, the hierarchical algorithm requires *countDB* to find the safe region and *listDB* to finally find all objects enclosed in the safe region. Due to the simplicity of the hierarchical expansion approach and its similarity to Algorithm 2, we do not list its pseudocode here.

$$t_{\text{hierarchical}} = O\left(\left(\frac{K}{n\delta^2} + \log_4 \frac{4K}{n\delta^2}\right) t_{\text{read}} + t_{\text{range}}\right) \tag{4}$$

3.3.3 Hilbert expansion

The progressive expansion strategy and in general similar linear expansion strategies such as the one proposed in [43], are usually very efficient in finding the safe region for the query q due to their simplicity. However, as noted in Sect. 3.1, the time it takes to find the region that includes at least K objects can be prohibitively large for non-uniform datasets (see Sect. 6). The Hilbert expansion overcomes this drawback by navigating in the *hilbDB* index which only stores cells that include at least one object. The main advantage of using *hilbDB* is that once the cell with closest H-value to $\nu(q)$ is found, expanding the search in either direction requires only $K - 1$ more private reads to generate a safe region. This 1-dimensional search gives a huge performance gain at the cost of generating a larger search region due to the asymmetric and 1-dimensional Hilbert expansion. In Sect. 6, we extensively study these trade-offs. Algorithm 3 illustrates the (simplified) Hilbert expansion approach. The search begins by computing the Hilbert value of q and identifying its immediate neighboring cells in the Hilbert space (lines 1–4). Next, these cells are privately read and objects within them are added to the result set (using *HasMoreObjects* and *NextPOI* functions) and the search expands in Hilbert space until K objects are found (lines 5–16). The rest is similar to Algorithm 2. Equation 5 denotes its complexity (the curve order is a very small number
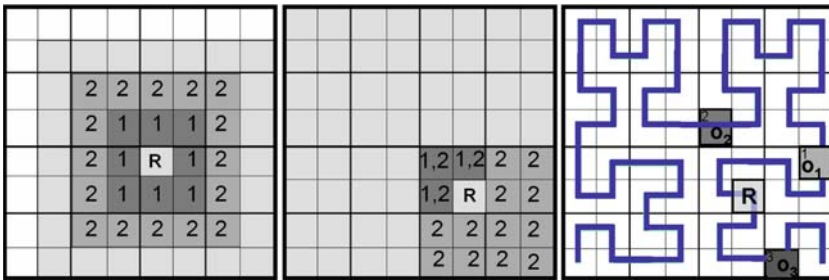
Fig. 4 Progressive, hierarchical and Hilbert KNN algorithms

($N < 10$)). The t_{Hilbert} consists of the time to compute $v(q)$, its closest c_{id}'s (accessing at most $K - 1$ other records) to construct the safe region, and finally performing a range query, respectively.

$$t_{\text{Hilbert}} = O\left(\log 2^{2N} + K t_{\text{read}} \log 2^{2N} + t_{\text{range}}\right) = O\left(K t_{\text{read}} + t_{\text{range}}\right) \tag{5}$$

Algorithm 3 KNN-Hilbert(K, q)

Require: K, q

$\quad c_{id} \leftarrow v(q.x, q.y)$;

2: $qIndexMore \leftarrow \min c_{id} | c_{id} \in hilbDB \ \& \ c_{id} \geq v(q.x, q.y)$;

$\quad G \leftarrow read(hilbDB, qIndexMore)$;

4: $qIndexLess = G.prev$;

$\quad cnt \leftarrow 0; S \leftarrow \emptyset$;

6: **while** $(cnt < K)$ **do**

$\quad\quad G \leftarrow read(hilbDB, qIndexMore)$;

8: $\quad$ **while** $(G.HasMoreObjects() \ \&\& \ (cnt < K))$ **do**

$\quad\quad\quad S = S \cup G.\text{NextPOI}()$;

10: $\quad\quad\quad cnt + +$;

$\quad\quad$ **end while**

12: $\quad qIndexMore = G.next$;

$\quad\quad G \leftarrow read(hilbDB, qIndexLess)$;

14: $\quad$ **while** $(G.HasMoreObjects() \ \&\& \ (cnt < K))$ **do**

$\quad\quad\quad S = S \cup G.\text{NextPOI}()$;

16: $\quad\quad\quad cnt + +$;

$\quad\quad$ **end while**

18: $\quad qIndexLess = G.prev$;

$\quad$ **end while**

20: $R' = safeRegion(S)$;

$\quad$ **return** $Range(R')$;

Figure 4 illustrates the three variants of evaluating KNN queries privately. The numbers in each cell show the step at which the cell is examined. For readability, cells examined in step 3 are shaded instead of being numbered in the first two images.

4 Optimizations

In this section, we first discuss how the granularity of the underlying grid can affect query processing and how the right granularity can be computed. Next, we discuss our secure padding scheme that while protecting data privacy, incurs up to 90% less overhead compared to a naive padding approach.

4.1 Grid granularity

Choosing the right value of δ can significantly improve the overall efficiency of the above algorithms. There is an obvious trade-off between two competing factors in choosing the right value of δ. As δ grows, a coarser grid (having less cells) decreases the total number of cell retrievals. This is desirable given the relatively high cost of each private read from the database. However, large cells result in retrieving more excessive (unneeded) objects which coexist in the cell being retrieved. These excessive objects result in higher computation and communication complexity which increase the overall response time. Similarly, small values of δ result in an inverse scenario. The following theorem shows how the optimal value of δ can be calculated offline (a simplified version of this theorem can be found in [43]).

Theorem 1 *For a grid of size $\frac{1}{\delta} \times \frac{1}{\delta}$ which contains n objects, $\delta \approx \sqrt[3]{\frac{t_c}{t_o}} \times \frac{1}{\sqrt{n}}$ is the optimal grid granularity to minimize the total query time for all three algorithms where $\frac{t_c}{t_o}$ denotes the relative time complexity of identifying a cell and querying it, compared to processing an object.*

Proof For all three algorithms, the overall query processing time T is dominated by the safe region (i.e., R') computation which is the time required to query n_c cells and to process n_o objects located in them, thus $T = t_c n_c + t_o n_o$. We show how the optimal value of δ can be derived for the progressive expansion strategy. The proof for other two algorithms is similar. Computing R' for a KNN query involves finding the circle $C(o, r)$, centered at point o with radius r, which includes the Kth nearest object to the query point. Therefore, $r \approx \sqrt{\frac{K}{\pi n}}$. A square S of $n_c = \frac{(2r+\delta)^2}{\delta^2}$ cells bounding C is guaranteed to contain the result set. S on average includes $n_o \approx (2r + \delta)^2 n$ objects. Replacing n_c and n_o in $T = t_c n_c + t_o n_o$ with the above values and setting $\frac{\partial T}{\partial \delta} = 0$ yields $\delta^3 = \frac{t_c r}{t_o n}$ or $\delta = \sqrt[3]{\frac{t_c}{t_o} \sqrt{\frac{K}{\pi}} \frac{1}{\sqrt{n}}}$.

For $K << n$, the above formula can be simplified to $\delta = \sqrt[3]{\frac{t_c}{t_o}} \frac{1}{\sqrt{n}}$ □

In Sect. 6, we empirically find δ using our measured values of $\frac{t_c}{t_o}$ and show that it is very close to the value calculated above.

4.2 Secure record decomposition and padding

A closer look at the $listDB$, $countDB$ and $hilbDB$ indexes, stored in an encrypted form at the untrusted server, reveals that the records of the $listDB$ and $hilbDB$ indexes do not have the same length and the size of each record is in fact determined by the object distribution in the 2-D space. The reason behind this difference is the fact that $countDB$ only stores aggregate object information as opposed to the other two indexes storing exact object locations. The unequal record sizes of $listDB$ and $hilbDB$ can result in several security vulnerabilities while using the PIR algorithm. This is because Algorithm 4 assumes that all database records
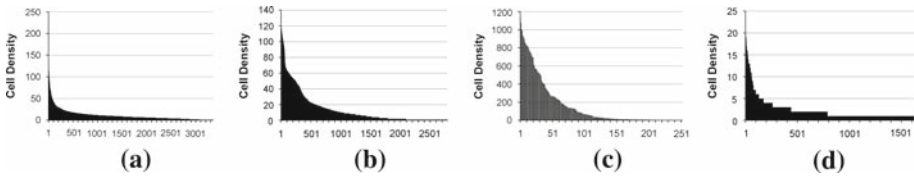
Fig. 5 Record size distribution. **a** Real-world; **b** uniform; **c** highly skewed; **d** sparse uniform
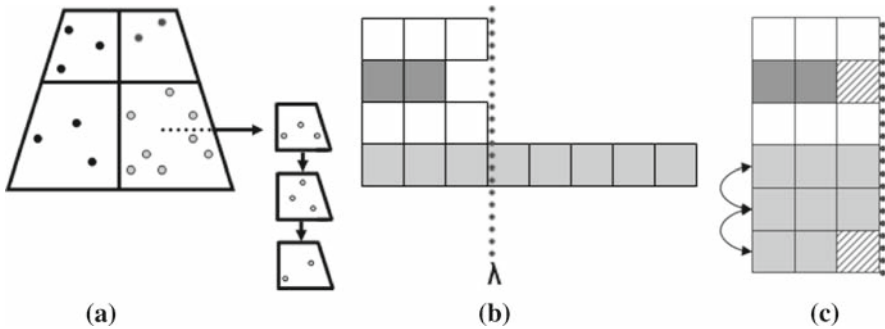

Fig. 6 Record decomposition and padding. **a** Object space; **b** database records; **c** padded records

have the same size. However, if the records have different lengths, it becomes possible for the server to identify a subset of records, with one of the records being the ciphertext of a plaintext known to the attacker, simply by correlating the length of the encrypted records and the length of the plaintext (e.g., using the publicly available set of all restaurants).

One obvious solution is to pad each record with junk data until each record size matches the length of the largest record. However, this results in an explosive growth in the size of the dataset. For instance, in our real-world dataset (refer to Sect. 6 for the characteristics of our four datasets), padding all the $listDB$ records results in a 1,700% increase in the size of the index. This is because the highly dense cells are vastly outnumbered by cells with many fewer objects (e.g., one cell containing over 200 objects vs. average object/cell density of around 10). Figure 5 illustrates the object density for each cell (represented by its c_{id}) for our experimental datasets. These graphs show the inefficiency of using a naive padding approach to secure data.

In order to overcome this drawback, we devise a *record decomposition and padding* scheme which first breaks records larger than a certain *cutoff threshold* to reduce the size of the largest record. The key advantage of this approach is that by choosing the right cutoff value, the space and complexity overhead of padding can be significantly reduced. Figure 6 illustrates how our scheme works. Given a cutoff threshold λ, and a record r whose size is larger than λ, we first recursively divide r into two smaller records r_1 and r_2 where $|r_1| = \lambda$. We refer to r as the original record and to r_1 and r_2 as its decomposed records. To maintain the logical correspondences among decomposed records, every record is assigned a pointer p which points to the index of its respective r_2 during the record decomposition and to an invalid dummy location otherwise (i.e., if $|r| \leq \lambda$). For each record, this process is recursively continued until $|r_2| \leq \lambda$. Next, all records smaller than λ are padded to make them equal in size. Figure 6a shows the object space, and its corresponding $listDB$ index is shown in Fig. 6b. Utilizing the above process, the index is first decomposed, the links are added and finally the smaller cells are padded as shown in Fig. 6c.

There are two important issues to be addressed using the decomposition technique discussed above. First, choosing the right cutoff threshold λ is of paramount importance. Note that $1 \leq \lambda \leq max(|r|)$ with $\lambda = max(|r|)$ representing the naive padding case which results to a huge increase in average record size. Also, $\lambda = 1$ represents decomposing every single record recursively such that we end up with each record storing only one object. Any value of λ in between these two extremes results in adding $n_{c'}$ more records as well as padding an equivalent of $n_{o'}$ objects (note that this padding does not actually add objects to the records, it instead adds dummy data at the end of each record whose size is always a multiple of an object's size). The overhead incurred by the padding can then be denoted by $Overhead = n'_c t_c + n'_o t_o$. While the first term makes PIR more costly (it retrieves more records privately for the same query), the second term negatively affects the client server communication cost (due to retrieving larger records for the same query). Therefore, based on the characteristics of the PIR technique and the overhead equation above, the right value of λ minimizing the overhead can be computed. In Sect. 6, we compare the overhead of our padding scheme with the naive padding technique.

With our proposed padding technique, a new information leak concern may arise due to adding pointers between decomposed records. Hence, we need to ensure that the pointers connecting different pieces of an original record do not leak any information to an adversary. This is important because while querying any original (and larger than cutoff threshold) record r, r_1 and r_2 will be requested sequentially. Fortunately, the $read()$ function discussed in Sect. 2.2 guarantees perfect secrecy regardless of the sequence of the records being queried. In other words, any PIR algorithm by design is resilient toward correlation attacks and thus the attacker cannot learn any information from the sequence of records being retrieved from the untrusted server. Therefore, while record encryption protects *content* (i.e., objects and pointers), PIR protects *access patterns* from leaking any sensitive information to adversaries. In Sect. 5, we elaborate on how an original record is queried by recursively retrieving its decomposed elements.

5 A sample implementation

In Sect. 2, we discussed the notion of privately retrieving an object from a database using the *read* operation. Here, we detail how we implemented such a protocol using an almost optimal PIR scheme proposed in [3,4] which utilizes *secure coprocessors* as a trusted platform to execute an efficient PIR protocol.

5.1 Hardware-Based PIR

A Secure Coprocessor (SC) is a general purpose computer designed to meet rigorous security requirements that assure unobservable and unmolested running of the code residing on it even in the physical presence of an adversary [34]. These devices are equipped with hardware cryptographic accelerators that enable efficient and fast implementation of cryptographic algorithms such as DES and RSA [32]. Recent advances in hardware technology have enabled successful implementation of several real-world applications such as data mining [6] and trusted web servers [22] on trusted computing environments. The use of secure coprocessors has also been proposed to increase the security of outsourced data and reduce the bandwidth requirements in the database as a service model [29]. In particular, two studies have designed and implemented privacy-enhancing features using IBM 4758 Secure Coprocessors. In [19], the authors implement a privacy enhanced X.509 certificate directory to

enhance client privacy while accessing server data. Similarly,the study by [20] details how IBM's secure coprocessors are used as trusted third parties to implement a secure function evaluation scheme.

Note that trusting a secure coprocessor is fundamentally different from trusting an anonymizer. This is due to the fact that while anonymization-based approaches reduce user query location exposure by blurring or making it indistinguishable, our PIR-based approach addresses an additional and orthogonal problem of query content privacy. In other words, regardless of the anonymization technique used, the server still infers sensitive information about querying location by monitoring the requested information. However, using our hardware-based PIR, the server is entirely blinded from learning any information from its communication with the secure coprocessor. Therefore, while both techniques need to rely on some entity to ensure privacy, anonymizers attempt to protect user privacy and still leak information through access patterns. However, in our approach secure coprocessors fully protect sensitive user location information by addressing both user (i.e., query) and access (i.e., response) privacy.

Trusting SC is also substantially different from trusting a location server in several respects. Aside from being built as a *tamper resistant* device, secure coprocessors are specifically programmed to perform a given task while location servers consist of a variety of applications using a shared memory. Therefore, unlike the secure coprocessor in which the users only have to trust the designer, using a location server requires users to trust the server admin and all applications running on it, as well as its designer. Last but not least, in our setting, the secure coprocessor is mainly a *computing* device that receives its necessary information, per session from the server, as opposed to a server which both stores location information and processes spatial queries.

The idea behind using SC is to place a trusted entity as close as possible to the untrusted host to disguise the selection of desired records within a black box. In order to avoid the linear cost of going through each record in the host or sending the entire dataset to the user (i.e., $O(n)$ computation and communication cost, respectively), we use the technique proposed by Asonov et al. [4] to achieve optimal (i.e., constant) query computation and communication complexity at the cost of performing as much offline precomputation as possible. Since [4] uses shuffling techniques, we first offer a brief overview of how shuffling can be efficiently performed.

Definition 5 *Random Permutation:* For a database DB of n items, the random permutation π transforms DB into DB_{π} such that $DB[i] = DB_{\pi}[\pi[i]]$.

For example for DB= $\{o_1, o_2, o_3\}$ and $DB_{\pi} = \{o_3, o_1, o_2\}$ the permutation π represents the mapping $\pi = \{2, 3, 1\}$. Therefore, $DB[1] = DB_{\pi}[\pi[1]] = DB_{\pi}[2] = o_1, DB[3] = DB_{\pi}[\pi[3]] = DB_{\pi}[1] = o_3$ etc. It is easy to verify that the minimum space required to store a permutation π of n records is $O(n \log n)$ bits.

The basic idea behind utilizing a secure coprocessor is to use π to privately shuffle and then encrypt the items of the entire dataset DB. While this encrypted shuffled dataset DB_{π} is written back to the server, SC keeps π for itself. Later, a user interested in the ith element of DB encrypts his query using SC's public key and sends it to SC through a secure channel. SC can then retrieve and decrypt $DB_{\pi}[\pi[i]]$, re-encrypt it with users' public key and send it back (hereinafter we distinguish between a *queried item* which is the item of interest requested by the user and *retrieved/read record* which is the item SC reads from DB_{π}). Although the server is *blindly* retrieving an encrypted record and returning it to SC, the scheme is not yet private. Launching a *chosen plaintext cryptanalysis*, the protocol still leaks to the untrusted dataset owner whether queries q and q' asked for the same item or not. In order to avoid this

problem SC maintains a list L which contains the indices of all items retrieved so far. SC also caches the records retrieved from the beginning of each session. In order to answer the kth query, SC first searches its cache. If the item does not exist in the cache, SC retrieves $DB_\pi[\pi[k]]$, stores it in its cache and adds k to L. However, if the element is already cached, it randomly reads a record not present in its cache and caches it. With this approach, each record of the database might be read at most once regardless of what items are being queried. This way, an adversary monitoring the database reads can obtain no information about the record being retrieved. Once each record is retrieved and decrypted, we check whether it points to another record in the database (this happens if a decomposed record is retrieved) and we recursively call the $read()$ function to retrieve the next record.

The problem with the above approach is that after $T_{\text{threshold}}$ retrievals, SC's cache becomes full. At this time a *reshuffling* is performed on DB_π which clears the cache and L. Note that since $T_{\text{threshold}}$ is a constant number independent of n, query computation and communication cost remain constant if several instances of reshuffled datasets are created offline[4], alternatively the shuffling can be performed regularly on the fly which makes the query processing complexity equal to the complexity of the recurring reshuffling which takes $O(n)$ for a database of size n [37]. Algorithm 4 details how the $read$ operation is performed privately. Note that between the shufflings, it reads a different record per query and thus ensures each record is accessed at most once.

We have now developed the necessary operations behind a *read* request formed as $read(DB_\pi, i)$. All details regarding the shuffling and the permutation are hidden from the entity interacting with DB_π. The average time it takes to privately retrieve a record from DB_π can be written as Eq. 6. We assume a single SC and no use of parallelism (if multi-threading is used in SC or if more than one SC is available, $t_{\text{read}} \approx 0$ since an unused permuted and encrypted instance of DB can always be made available). Note that t_{read} is inversely proportional to a linear increase in $T_{\text{threshold}}$.

$$t_{\text{read}} = O\left(\frac{n}{T_{\text{threshold}}}\right) \tag{6}$$

Theorem 2 *Algorithm 4 does not leak information.*

Proof Proved by Asonov [3] and Wang et al. [37] showing the joint entropy of a sequence of queries is maximal. □

During the execution of the *read* operation, we might reach $T_{\text{threshold}}$, the threshold of maximum allowed elements to be retrieved before shuffling (Sect. 2.2) in which case either swapping to an unused instance of DB_π is required or reshuffling is performed online to generate a new instance before the range and KNN algorithms can continue.

We can now discuss how the PIR scheme (Sect. 5.1) and private spatial queries (Sect. 3) are integrated. The entire query processing can be divided into the following phases.

5.2 Preprocessing

During this phase, the server first creates *listDB*, *countDB* or *hilbDB* index (while *listDB* is required by range queries, any of the above indexing schemes can be chosen depending on the KNN algorithm of choice). To simplify notation, in this section we denote them as DB_1, DB_2 and DB_3, respectively. Next, SC generates a random permutation π and privately shuffles the above indexes and encrypts them. The encrypted shuffled databases $DB_{\pi 1}$, $DB_{\pi 2}$ and $DB_{\pi 3}$ are then written back to the server. Note that depending on the storage availability of

---

Algorithm 4 $read(DB_\pi, i)$

Require: DB_π, T {Threshold}, L {Retrieved Items}

 1: **if** $(|L| \geq T)$ **then**
 2:   $DB_\pi \leftarrow$ Reshuffle DB_π using a new random permutation π;
 3:   $L \leftarrow \emptyset$;
 4:   Clear SC's cache
 5: **end if**
 6: **if** $i \notin L$ **then**
 7:   $record \leftarrow DB_\pi[\pi[i]]$;
 8:   Add $record$ to SC's cache
 9:   $L = L \cup \{i\}$;
10:   **if** $valid(record.p)$ **then**
11:     $read(DB_\pi, p)$;
12:   **end if**
13: **else**
14:   Read $record$ from SC's cache
15:   **if** $valid(record.p)$ **then**
16:     $read(DB_\pi, p)$;
17:   **end if**
18:   $r \leftarrow$ random index from $DB_\pi \setminus L$;
19:   $temp \leftarrow DB_\pi[\pi[r]]$;
20:   Add $temp$ to SC's cache
21:   $L = L \cup \{r\}$;
22: **end if**
23: return $record$;

SC, several instances of $DB_{\pi i}$ can be created offline and stored at the server as long as their corresponding permutation indices are kept in SC.

The server might try not to follow the protocol by manipulating the construction of the permuted databases. However, Theorem 2 states that the server cannot *infer* any patterns by monitoring retrievals. Also, any diversion from the protocol is quickly noticed by the users who will abandon the protocol immediately.

5.3 Query processing

During the query processing phase, users first establish a secure channel with SC through an SSL tunnel and submit their queries to SC. A query q initiated by a user u_i is received by SC as $Enc_{spk}(q, s_{id})$ where q is one of the algorithms discussed in Sect. 3 and s_{id} is the session id for the communication between SC and the user. Using Enc, each user encrypts his query with SC's public key spk and sends it along with his s_{id}. Depending on the value of q, SC invokes one of the algorithms discussed in Sect. 3. While executing the queries, each SC's interaction with the server is via the private $read()$ requests by which SC privately retrieves a sequence of encrypted records from $DB_{\pi 1}$, $DB_{\pi 2}$ or $DB_{\pi 3}$ (depending on the type of query and KNN processing algorithm used) that contains the result set for q. Each member of the result set is first decrypted by SC and then re-encrypted with the user's public key and the compiled result set is transferred to the user (to thwart replay attacks, SC binds a nonce to the result set). Note that the result set R' returned by each algorithm might include some extra objects. Although SC can remove them before sending the final results to the user,
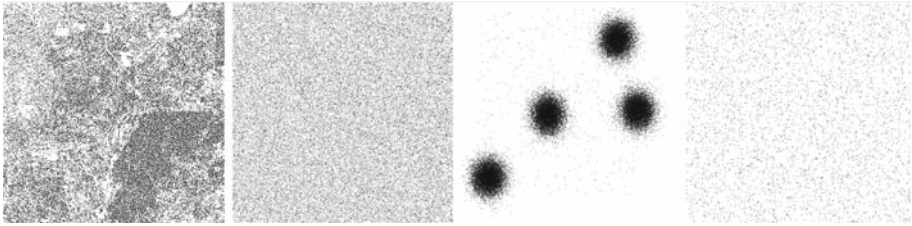
Fig. 7 Real-world, uniform, highly skewed and sparse datasets

we assume the filtering step is performed by the user mainly to reduce SC's computation overhead.

The main success factor behind this technique is the *anonymization* of user queries by converting them into a sequence of PIR reads from the server all being performed by the secure coprocessor. Therefore, the server is no longer able to trace database records back to separate user queries. In other words, by breaking the correspondence between user queries and database reads, and detaching user identities from their queries (through SC) the server cannot infer any information about the query (e.g., size of R or value of K). This is critical to hinder adversaries from learning user locations by gaining information about their query parameters. Note that a conventional PIR scheme would have to blindly query the entire database for each grid cell being queried. Our main motivation behind designing the private index structures in Sect. 3.1 is to avoid this processing cost.

Theorem 3 *Using the above framework, range and KNN queries are blindly evaluated.*

Proof We need to prove that while evaluating the query q, $P_q(u_j) = \frac{1}{m}$ and $P'_q(l_i) = \frac{1}{area(A)}$ (see Definitions 1 and 2). Since SC is the only entity interacting with the server, no user identity information is passed to the server and therefore from server's point of view, the query could have been initiated by any u_j for $j \in \{1 \ldots m\}$ which means $P_q(u_j) = \frac{1}{m}$. Furthermore, according to the query processing logic, SC decomposes q into a set of private $read()$ requests for the records (cells) $r_1, r_2, r_3, \ldots, r_s$ from the server's encrypted and shuffled databases. Note that r_i's are highly correlated with q and $read(DB_\pi, i)$ operations are monitored by the sever. However, Theorem 2 guarantees $read(DB_\pi, i)$ leaks no information about i, r_i and thus about q. Therefore, no location information is revealed to the server through cell retrievals. Hence, $P'_q(l_i) = \frac{1}{area(A)}$ □.

6 Performance evaluation

In this section we empirically examine the overall efficiency of our framework. We conducted extensive experiments to determine the effectiveness of our framework in terms of (1) the effect of system parameters such as the grid size, the cache size, etc. (2) the overall response time of the range and three KNN algorithms for different datasets and (3) the overall end-to-end performance of the framework.

6.1 Experimental setup

Our experiments are performed on four different datasets (Fig. 7); three of which contain around 40, 000 objects. These datasets are as follows: (a) a real-world dataset obtained from

NAVTEQ (www.navteq.com) covering the restaurants in an 1,280 by 1,280 km area in central United States. (b) a uniform distribution (c) a synthesized highly 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 and (d) a sparse dataset of 4, 000 uniformly distributed objects. Experiments were run on an Intel P4 3.20 GHz with 2 GB of RAM emulating a secure coprocessor.

One of the key challenges in hardware-based PIR is dealing with relatively low computation power and available storage space of secure coprocessors. For example, the IBM's PCIXCC secure coprocessor runs at 266 MHz and supports 80 MB of main memory, Ethernet connection and a Linux OS [2]. Although such processors are slower than their non-secure counterparts, they are equipped with cryptographic accelerators that significantly outperform a typical high-speed processor in performing cryptographic operations. The most recent IBM 4764 PCI-X Cryptographic Coprocessor [36] generates up to 4702048-bit RSA signatures per second compared to a P4 @3.4 GHz that generates around 40 signatures during the same time (i.e., more than one order of magnitude improvement). Due to very frequent encryption (decryption) operations required in our algorithms, such accelerators can significantly improve the overall response time. Furthermore, as we show in this section, for optimal system parameters (i.e., grid, cutoff threshold and cache size), our response times are mostly in the orders of milliseconds and even a secure coprocessor that runs almost 10 times slower, still achieves very satisfactory response times. With regard to space, running range and KNN algorithms on SC does not strain its relatively limited memory due to the fact that the large index structures are being stored at the untrusted server and the proposed range and KNN algorithms are designed to consume the limited amount of space available in SC. In Sect. 3.1, we discussed why we chose to use regular grids as opposed to other indexing variants such as r-trees or kd-trees. In addition, here we observe that using such techniques require the tree information to be either stored at SC or privately and incrementally queried from the server per query. While the former approach is infeasible due to highly restricted available storage and computation resources of SC, the latter approach also incurs significantly more query processing costs due to private retrieval of tree nodes prior to retrieving the actual query result set from the server.

6.2 Space complexity analysis

As our first set of experiments, we briefly analyze the space requirements of our entire framework. Table 1 summarizes the required space for storing each item in L (the list storing the index of previously retrieved items in PIR), SC's cache and the permutations π and π', respectively. The second column shows the exact values for $n = 10^5$ (i.e., the total number of objects), $M = \frac{1}{\delta} = 2^7$, $T_{\text{threshold}} = 10^3$ and $s = sizeOf(double)$ in bits. These values closely represent the bulk of our experiments. L represents the number of bits needed to store each c_{id} of an $M \times M$ grid. For each c_{id}, the cache stores on average $\frac{n}{M^2}$ objects each represented by the triplet $\langle longitude, latitude, id \rangle$. As expected, the most significant space requirement is imposed by π and π' where each of the M^2 rows in π stores a mapping of one c_{id} to another. However, as discussed in Sect. 3.1, their sizes are determined by the granularity of the underlying grid structure and are invariant of n (as we are indexing the cells instead of the objects located in them). We note that since our records are relatively small, the storage requirements of SC's cache are fairly nominal. However, if enough space is not available in SC's cache to store significantly larger records, we can use a variant of PIR scheme proposed by [3] which does not require caching at all by reading all $k - 1$ previously

Table 1 Storage requirements of SC

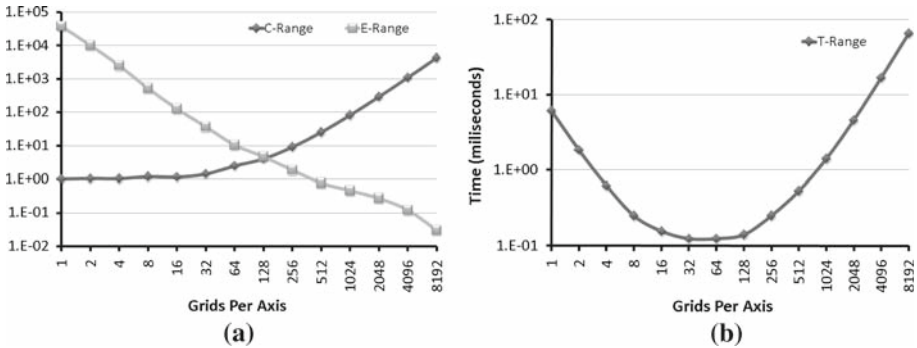|	Storage (byte)	Value (Kbyte)
L	$\frac{\log(M)}{4}$	≈ 2
Cache	$\frac{\log(M)}{4} + \left(\frac{2 \times s + \log(n)}{8}\right) \times \frac{n}{M^2}$	≈ 26
π (π')	$M^2 \log(M)$	≈ 230

Fig. 8 Effect of δ on range algorithm for the real-world dataset. **a** C and E; **b** time

accessed records for evaluating the kth query instead of using its cache to read only a single element.

6.3 The effect of δ

As we discussed in Sect. 3, choosing the right value of δ can significantly affect query performance. In this section, we evaluate how the performance of our proposed algorithms changes for different values of δ. To focus on the overhead of our range and KNN algorithms, for now we assume database reads are not private (i.e., $t_{\text{read}} \approx 0$). Later in Sect. 6.7, we replace database reads with the private *read* function developed in Sect. 2.2 to study the overall query response time which also includes the PIR overhead.

Unless otherwise stated, the results of each experiment is averaged over 200 randomly generated range queries of size $100 \, \text{km}^2$ and KNN queries with $K = 100$ (these are much larger than the typically used range and KNN queries in many LBS). For each experiment, we measure (a) the average number of cells (i.e, records) being queried from any of our private index structures, hereafter being represented by C (b) the average number of excessive objects being queried (i.e., $|R'| - |R|$ for range and $|R'| - K$ for KNN queries) hereafter represented by E and (c) the overall query response time T in milliseconds. Figure 8 illustrates the effect of δ on these three parameters for the range queries, and Fig. 9 illustrates how these values change in three KNN query processing algorithms (note that the Y-axes numbers are in logarithmic scale). All above experiments are performed on our real-world dataset. As expected, using the uniform dataset, the results demonstrated very similar trends to the real-world data (both for the above experiments as well as the rest of our experiments in this section). Therefore, we only report the results for the three remaining datasets.

Obviously, it is desirable to minimize all three factors C, E and T simultaneously, to achieve the optimal performance. However, as discussed before, small values of C are
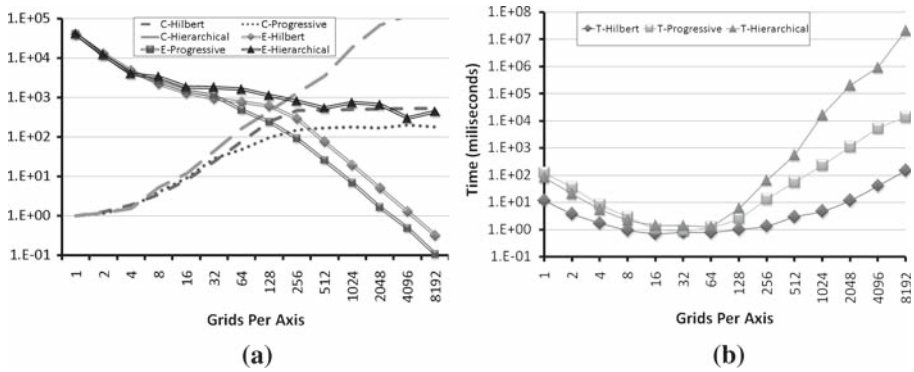
Fig. 9 Effect of δ on progressive, Hilbert and hierarchical algorithms for the real-world dataset. **a** C and E; **b** time

achieved with very coarse grids which clearly increases E (i.e., false positives) and hence the overall response time for both range and KNN query processing. Alternatively, a small value of E can only be guaranteed when dealing with very fine-grained cells (i.e., small values of δ) which comes at the cost of a quadratic increase in t_{range} and also a quadratic increase in the $\frac{K}{n\delta^2} \times t_{\text{read}}$ and t_{range} terms of $t_{\text{progressive}}$. However, shrinking δ only increases the t_{range} term for the Hilbert Algorithm and hence incurs up to two orders of magnitude less overall response time compared to the progressive algorithm (see the derivations of t_{range}, $t_{\text{progressive}}$ and t_{Hilbert} in Sect. 3.3). Furthermore, observe that as discussed in Sect. 3.3.2, the exponential growth of the examined region in KNN-Hierarchical results in a huge increase for C and E thus taking significantly longer to execute KNN queries compared to the other two algorithms. Therefore, for the rest of our experiments, we only report the efficiency of the range and the two superior KNN Algorithms 2 and 3.

Observe that if $\frac{t_c}{t_o} = 1$, the value of δ at intersection of C and E would correspond to T's global minimum. However, for our experimental setup, we obtained $\frac{t_c}{t_o} \approx 35$. Utilizing Theorem 1, $\frac{1}{\delta} \approx 61$ for $n = 40,000$. As Figs. 8 and 9 illustrate, the measured values of T confirm our analytical derivation of optimal δ (i.e., $\frac{1}{\text{grids per axis}}$). Hence, δ can be chosen in advance using Theorem 1. When dealing with our first three datasets, we set $\delta = \frac{1}{64}$ for our remaining experiments.

In our next sets of experiments, we evaluated the performance of the progressive and Hilbert algorithms on highly skewed and sparse datasets. Figure 10a, b illustrates the results for the skewed dataset. Although T increases in both algorithms, Algorithm 3 still significantly outperforms Algorithm 2. Also, the average number of cells queried by progressive (compared to the real-world dataset) grows twice faster than the Hilbert algorithm. This confirms the superiority of Hilbert indexing for non-uniform datasets. Figure 10c, d shows how our KNN algorithms perform on the sparse dataset. Setting $n = 4,000$ in Theorem 1 yields $\delta \approx \frac{1}{19}$ which conforms to our empirical local minimum of T at $\delta = \frac{1}{16}$. Furthermore, compared to the real-world dataset, the progressive and Hilbert algorithms each incurs 15% and 10% increase in C and one order of magnitude increase in T, respectively. This is expected given the derivations of $t_{\text{progressive}}$ and t_{Hilbert} from Sect. 3.3. Performing the same experiments for Algorithm 1 demonstrates similar behavior which is not shown here.
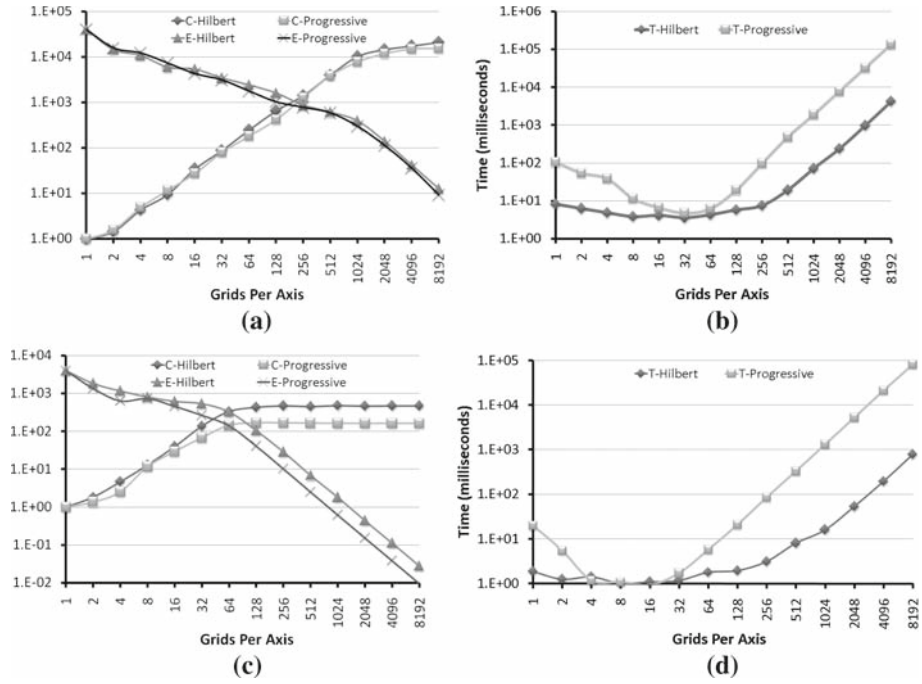
Fig. 10 Effect of δ for skewed (*top*) and sparse (*bottom*) datasets. **a** C and E; **b** time; **c** C and E; **d** time

6.4 Choosing the optimum cutoff value

Using the values of t_o and t_c derived in Sect. 6.3, we proceed to compute the optimal value of λ for each distribution. Figure 11 illustrates the padding overhead for different values of λ. It also illustrates how our secure padding scheme reduces the padding overhead compared with the naive padding without compromising security. For each distribution, we choose the λ that results in smallest padding overhead. For instance, setting $\lambda = 18$ results in a 90% reduction in cost of padding compared to the naive padding approach for our real-world dataset.

6.5 The effect of K

We proceed to examine the effect of K on KNN query processing time T. As Fig. 12 illustrates, values of C, E and T linearly increase with K for both algorithms. As expected, the Hilbert method outperforms progressive for all values of K (see Sect. 3.1).

6.6 The effect of Hilbert curve order

As our next set of experiments, we analyze how our Hilbert-based KNN Algorithm behaves for different values of the Hilbert curve order (i.e., N). Figure 13a shows how an increase in N quickly reduces the value of C which shows the curve's effect in more effectively finding adjacent objects to the query point. However, as the granularity of the Hilbert curve moves beyond the granularity of the underlying grid cell, the Hilbert nearest neighbor approximation reaches its maximum accuracy and stays constant. Similarly, an increase in N quickly reduces the response time due to the smaller size of the safe region. However, a further increase in
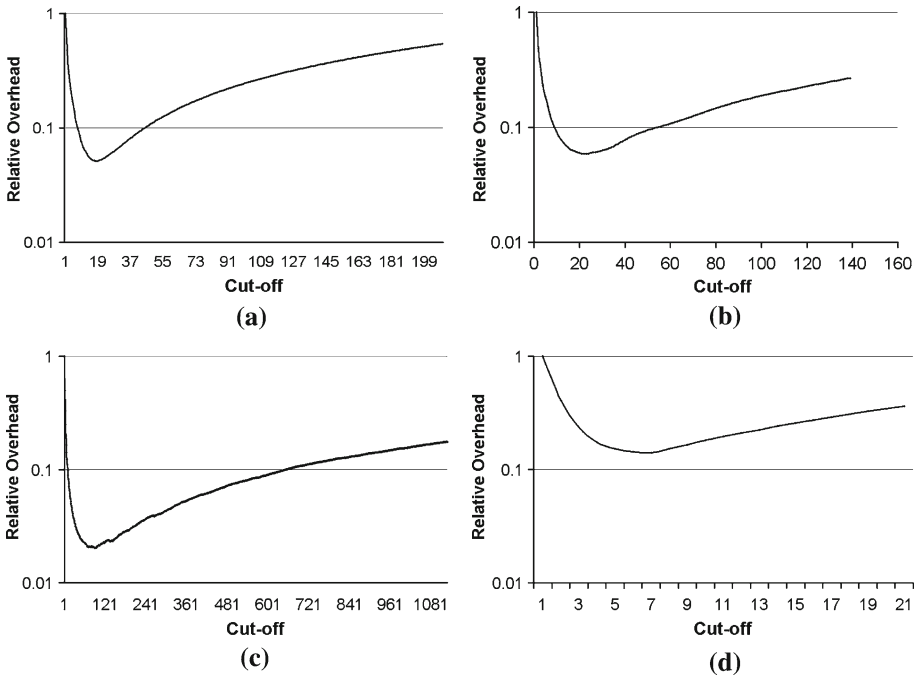
Fig. 11 Relative overhead reduction of secure padding for datasets of Fig. 7. **a** 90% (λ= 18); **b** 78% (λ = 23); **c** 89% (λ = 88); **d** 61% (λ = 7)
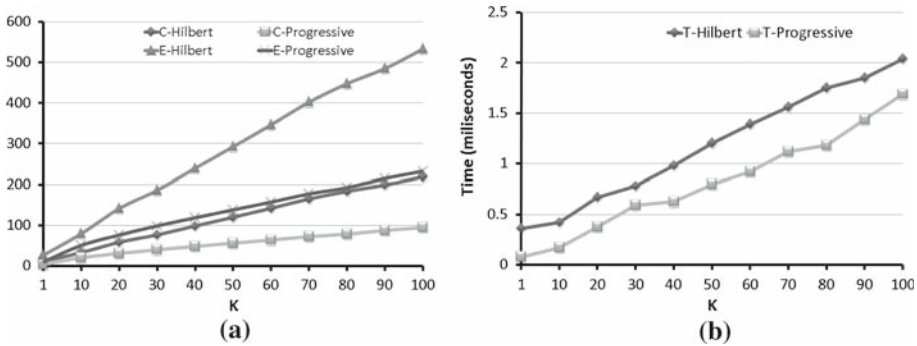

Fig. 12 Effect of K for the real world dataset. **a** C and E; **b** time

N increases the $K t_{\text{read}} \log 2^{2N}$ term in t_{Hilbert} up to the point where $2 \log 2^{2N}$ is dominated by the t_{range} term in overall query processing time.

6.7 End-to-End performance

So far we focused on the performance of our private spatial query processing schemes in the absence of PIR. We now measure the overall response time of our framework which includes the overhead t_{pir} introduced by Algorithm 4 for private evaluation of the above queries. Figure 14a–d illustrate the effect of PIR overhead on the end-to-end performance for 1, 000 randomly distributed range and KNN queries, respectively. Note that response time
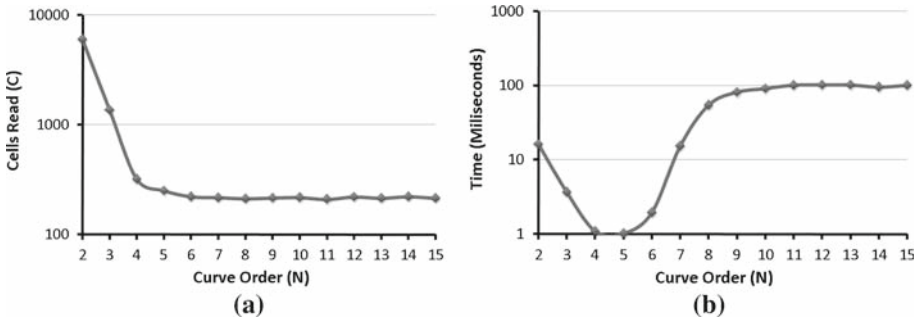
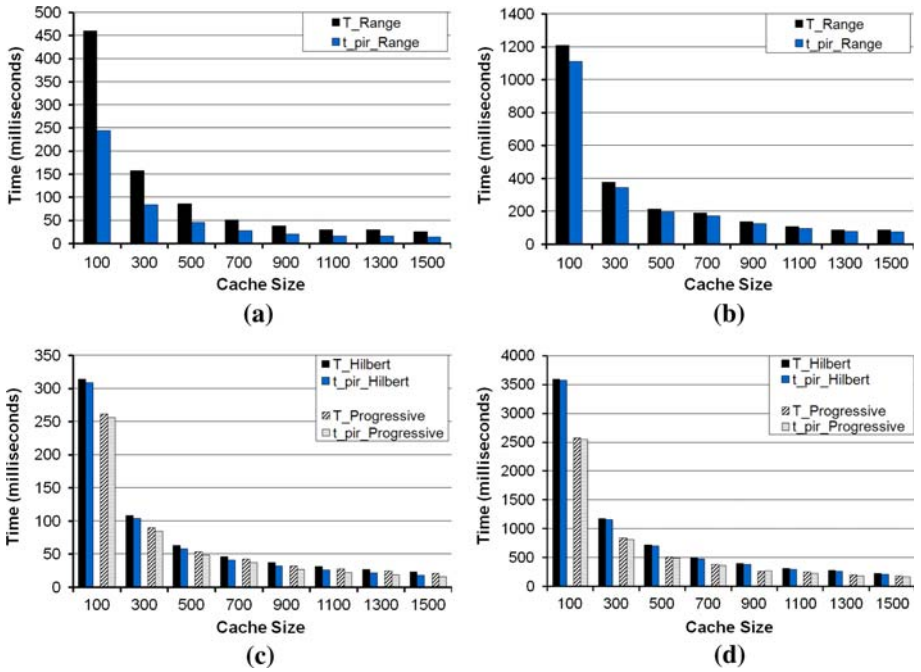**Fig. 13** Effect of N on C (*left*) and time (*right*) for the real world dataset using algorithm 3. **a** C; **b** time

Fig. 14 End-to-end performance for range (**a, b**) and KNN (**c, d**) Algorithms. (**a**) $\delta = 64$; (**b**) $\delta = 128$; (**c**) $\delta = 64$; (**d**) $\delta = 128$

is inversely proportional to the cache size (represented in terms of number of objects) and thus larger cache size reduces the overall query processing time due to less number of private reads and less frequent reshuffling. Figure 14 also shows how the overall query processing time is dominated by the relatively expensive PIR modules for all three algorithms.

The superior performance of Algorithm 2 over Algorithm 3 might initially look counter intuitive. This is because Fig. 9b showed how Hilbert-based KNN query processing outperformed the progressive expansion in the absence of PIR. While Algorithm 3 very efficiently computes the safe region, Algorithm 2 spends significantly more time to compute its (slightly smaller) safe region which includes fewer cells. However, adding PIR greatly increases the time to retrieve a cell due to the cost of reshuffling. Therefore, the cost of reading extra cells

	Progressive	Hilbert	exactNN
Communication Cost (bits)	630	1740	15200
Time (milliseconds)	166	187	33000
E (objects)	10	27	15

(a)
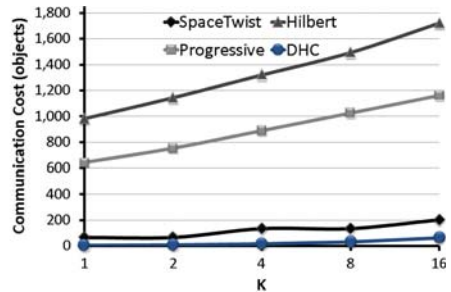
(b)

Fig. 15 Comparing with other approaches. **a** PIR-based; **b** transformation-based

is significantly higher than the non-PIR case. A careful examination of Fig. 9a reveals that Algorithm 3 retrieves more cells compared to Algorithm 2 which negatively affects t_{Hilbert} compared to $t_{\text{progressive}}$. In other words, the shuffling time imposed by the PIR routine dominates the savings in quick identification of the safe region by Algorithm 3. We finally note that if approximate results were needed, no safe region had to be computed and Algorithm 3 would still significantly outperform Algorithm 2.

6.8 Comparing with other approaches

In Sect. 5, we analytically showed the superiority of our approach in satisfying significantly more stringent privacy guarantees compared to the cloaking/anonymity-based approaches. However, an empirical comparison between our techniques and these studies is not straightforward because anonymity and cloaking approaches mostly evaluate performance based on the size of the K-anonymity set or the cloaked region and the effectiveness of the anonymization techniques used. More importantly, contrary to our approach, the level of privacy attained through anonymization highly depends on the number of users as well as their distribution. Therefore, in the following two sections, we only consider the PIR and transformation-based approaches for an empirical comparison.

6.8.1 PIR-Based approaches

We proceed to compare our approach with the recent work of Ghinita et al. [14] which similar to us utilizes PIR to evaluate 1NN queries while ensuring perfect privacy. Figure 15a summarizes the differences between our progressive and Hilbert algorithms with the exact NN algorithm proposed by [14] in terms of the communication and computation cost, as well as the number of excessive objects disclosed to users. The results show that our algorithms outperform the proposed exactNN algorithm in the amount of communication and computation. This is because by avoiding a secure coprocessor, the theoretical PIR protocol used in [14] incurs significantly more computation cost compared to hardware-based PIR techniques. Moreover, in order to provide a fair comparison, we reduced our CPU clock to a tenth of its original value to simulate the slower SC speed and used very moderate cache (32 Kb) with $M = \frac{1}{64}$. Finally, all three methods roughly disclose the same number of extra objects to clients while evaluating 1NN queries. Note that the exactNN algorithm proposed in [14] is only applicable for $K = 1$ and thus we were unable to further compare our approach with [14] for range and for $K > 1$ in case of KNN queries.

6.8.2 Transformation-based approaches

We also compare our work against the two transformation-based approaches discussed in Sect. 7. We compare the communication cost and privacy guarantees of our approach with that of [42] and the *Dual Hilbert Curve* technique of [25] denoted by DHC. Since neither approach provides the end-to-end computation cost, we were unable to compare the three approaches based on query processing time. However, we anticipate our approach to be clearly more computationally intensive due to the high cost of private record retrieval.

Similar to SpaceTwist, we used the dataset of 172, 188 school locations from USGS[2] to compare our approach against SpaceTwist and DHC. We generated 100 random KNN queries and studied the effect of varying K on communication cost measured in terms of the total number of points transmitted to the client. Our observations are illustrated in Fig. 15b. Among all four techniques, larger values of K linearly increase the communication cost. Although the communication costs of our approaches are comparable, DHC and SpaceTwist both outperform our PIR-based approaches. However, it is important to note that this is achieved at the cost of generating approximate response and leaking query location information (see Sect. 7). In fact for both techniques, we used their recommended parameter settings to optimize performance (vs. privacy or result accuracy). For instance, requiring SpaceTwist to return exact results in the above experiment results in a ninetyfold increase in the size of the result set.

7 Related work

In this section we review three classes of approaches proposed to enable location privacy in location-based services.

Location Anonymity and Cloaking. Inspired by anonymization techniques in privacy-preserving data mining, a large body of work in location privacy is based on the concept of K-anonymity or location cloaking [5,12,16,24,28]. With this approach, a trusted *anonymizer* blurs raw user locations by (for example) extending them from a *point* location to an *area* (spatial extent) and sending a region containing several other users to the untrusted server. However, aside from the well-known privacy issues of anonymization in data mining [31,35,38], location anonymization and cloaking suffer from several drawbacks. First, the users have to trust the anonymizer which is as sophisticated as the location server itself. More importantly, there are certain scenarios in which the private location information of users leak to malicious entities [24] or the cloaking process fails for certain user distributions or privacy preferences [5]. Furthermore, the quality of service or overall system performance degrades significantly as users choose to have more strict privacy preferences.

To alleviate the drawbacks of centralized cloaking, some studies propose a decentralized approach in constructing the cloaking region. The most notable work is proposed by [13] which utilizes a hierarchical overlay network resembling a distributed B+ tree for constructing the cloaked region. However, aside from very slow response time, such approaches assume all users trust one another and can communicate with each other in real time to construct the cloaking region both of which are impractical assumptions for real-world scenarios.

Query Transformation. A relatively new class of *transformation-based* approaches, avoid some of the shortcomings of cloaking-based techniques. Most recently, [42] proposed a framework termed SpaceTwist to blind an untrusted location server by incrementally retrieving points of interest based on their ascending distance from a transformed query

---

[2] http://geonames.usgs.gov/index.html.

point termed the *anchor* point which is a fake location near the query point. In another study, [25] propose *encoding* the query point and the object space to a Hilbert space using a one-way transformation and evaluating the query in the transformed space to preserve privacy. We note that while these two approaches mitigate some of the shortcomings of anonymity and cloaking-based approaches, they introduce a new set of drawbacks. First, both approaches are only practical when *approximate* results are desired as providing exact answers results in prohibitively high communication cost in [25] and severe query location information leakage in [42]. Furthermore, both approaches rely on a query processing technique only suitable for proximity queries such as KNN and do not discuss range queries. Finally, achieving perfect secrecy in the absence of PIR is an open problem even when approximate answers are sufficient and these two approaches are not exceptions in this sense.

Private Information Retrieval. The use of PIR to enable location privacy is also proposed by two other studies. Ghinita et al. [14] utilize computational PIR to enable private evaluation of first nearest neighbor queries. Although this work does not extend to KNN or range queries, we have compared our framework with [14] for $1NN$ queries in Sect. 6. Due to the hardware independence, the PIR protocol used in [14] incurs much more processing penalty than our hardware-based PIR approach. In fact, processing each private read requires a linear scan of database items at the server. Furthermore, the underlying PIR scheme incurs very costly communication complexity for each object retrieval. Also, Hengartner [17] presents an architecture that uses PIR and trusted computing to protect user locations from an untrusted server. However, the proposed architecture is not yet implemented.

8 Conclusion and future work

In this paper, we proposed a framework to evaluate private location-dependent queries utilizing the techniques developed in the private information retrieval literature. Using PIR, we satisfy more stringent user privacy guarantees compared to anonymity and cloaking-based approaches by blinding the server from learning any information about user locations as well as the content of their queries. We carried out extensive sets of experiments to empirically verify the effectiveness of our proposed approach. As part our future work, we plan to study novel techniques for enabling location privacy which are less costly than PIR-based techniques while still providing the same level of privacy.

References

1. Al-Muhtadi J, Campbell RH, Kapadia A, Mickunas MD, Yi S (2002) Routing through the mist: privacy preserving communication in ubiquitous computing environments. In: ICDCS'02, Austria, pp 74–83
2. Arnold TW, van Doorn L (2004) The IBM PCIXCC: a new cryptographic coprocessor for the IBM eServer. IBM J Res Dev 48(3–4):475–488
3. Asonov D (2004) Querying databases privately: a new approach to private information retrieval, vol 3128. Lecture notes in computer science. Springer, Berlin
4. Asonov D, Freytag JC (2002) Almost optimal private information retrieval. In: PET'02, San Francisco, CA, pp 209–223

5. Bamba B, Liu L, Pesti P, Wang T (2008) Supporting anonymous location queries in mobile environments with privacygrid. In: WWW'08, Beijin, China, pp 237–246
6. Bhattacharjee B, Abe N, Goldman K, Zadrozny B, Chillakuru VR, del Carpio M, Apte C (2006) Using secure coprocessors for privacy preserving collaborative data mining and analysis. In: DaMoN'06, Chicago, IL, p 1
7. Bouganim L, Pucheral P (2002) Chip-secured data access: confidential data on untrusted servers. In: VLDB'02, Hong Kong, China, pp 131–142
8. Cabbies threaten strike over GPS systems. http://www.cnn.com/2007/TECH/08/01/gps.taxi.strike.ap/index.html
9. Chor B, Kushilevitz E, Goldreich O, Sudan M (1998) Private information retrieval. J ACM 45(6):965–981
10. Damiani E, Vimercati SDC, Jajodia S, Paraboschi S, Samarati P (2003) Balancing confidentiality and efficiency in untrusted relational DBMSs. In: CCS'03, Washingtion, DC, pp 93–102
11. Faloutsos C, Roseman S (1989) Fractals for secondary key retrieval. In: PDS'89, New York, NY, pp 247–252
12. Gedik B, Liu L (2005) A customizable *k*-anonymity model for protecting location privacy. In: ICDS'05, Columbus, OH, pp 620–629
13. Ghinita G, Kalnis P, Skiadopoulos S (2007) PRIVE: anonymous location-based queries in distributed mobile systems. In: WWW'07, Alberta CA, pp 371–380
14. 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, pp 121–132
15. Gonzalez MC, Hidalgo CA, Barabasi A (2008) Understanding individual human mobility patterns. Nature 453:779–782
16. Gruteser M, Grunwald D (2003) Anonymous usage of location-based services through spatial and temporal cloaking. In: MobiSys'03, San Francisco, CA, pp 31–42
17. Hengartner U (2007) Hiding location information from location-based services. In: MDM'07, Mannheim, Germany, pp 268–272
18. Iliev A, Smith SW, (2004) Private information storage with logarithm-space secure hardware. In: International information security workshops, Toulouse, France, pp 201–216
19. Iliev A, Smith S (2005a) Protecting client privacy with trusted computing at the server. IEEE Secur Priv 3(2):20–28
20. Iliev A, Smith S (2005b) More efficient secure function evaluation using tiny trusted third parties. In: TR2005-551
21. Indyk P, Woodruff DP (2006) Polylogarithmic private approximations and efficient matching. In: TCC'06, New York, NY, pp 245–264
22. Jiang S, Smith S, Minami K (2001) Securing web servers against insider attack. In: ACSAC'01, Washington, DC, pp 265
23. Kalashnikov DV, Prabhakar S, Hambrusch SE (2004) Main memory evaluation of monitoring queries over moving objects. Distrib Parallel Databases 15(2):117–135
24. Kalnis P, Ghinita G, Mouratidis K, Papadias D (2006) Preserving anonymity in location based services. A technical report
25. Khoshgozaran A, Shahabi C (2007) Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In: SSTD'07, Boston, MA, pp 239–257
26. Khoshgozaran A, Shirani-Mehr H, Shahabi C (2008) SPIRAL, a scalable private information retrieval approach to location privacy. In: The 2nd international workshop on privacy-aware location-based mobile services (PALMS). In conjunction with MDM'08, Beijing, China
27. Kushilevitz E, Ostrovsky R (1997) Replication is not needed: single database, computationally private information retrieval. In: FOCS'97, Miami Beach, Florida, pp 364–373
28. Mokbel MF, Chow C-Y, Aref WG (2006)The new casper: query processing for location services without compromising privacy. In: VLDB'06, Seoul, Korea, pp 763–774
29. Mykletun E, Tsudik G (2005) Incorporating a secure coprocessor in the database-as-a-service model. In: IWIAć605, College Park, MD, pp 38–44
30. Online data gets personal: cell phone records for sale. http://www.washingtonpost.com/wpdyn/content/article/2005/07/07/AR2005070701862_pf.html
31. Qiu L, Li Y, Wu X (2008) Protecting business intelligence and customer privacy while outsourcing data mining tasks. Knowl Inf Syst 17(1):99–120
32. Sion R, Carbunar B (2007) On the computational practicality of private information retrieval. In: NDSS'07, San Diego, CA
33. Smith S (1996) Secure coprocessing applications and research issues. Los Alamos unclassified release LAUR -96-2805, Los Alamos National Laboratory

34. Smith SW, Safford D (2000) Practical private information retrieval with secure coprocessors. Technical report, IBM
35. Teng Z, Du W (2009) A hybrid multi-group approach for privacy-preserving data mining. Knowl Inf Syst 19(2):133–157
36. The IBM 4764 PCI-X cryptographic coprocessor, (April 2008). http://www-03.ibm.com/security/cryptocards/pcixcc/overperformance.shtml
37. Wang S, Ding X, Deng RH, Bao F (2006) Private information retrieval using trusted hardware. In: ESORICS'06, Germany, pp 49–64
38. Wang K, Fung BCM, Yu PS (2007) Handicapping attacker's confidence: an alternative to *k*-anonymization. Knowl Inf Syst 11(3):345–368
39. Warrior J, McHenry E, McGee K (2003) They know where you are. IEEE Spectr 40(7):20–25
40. Wireless location privacy: law and policy in the U.S., EU and Japan. http://www.isoc.org/briefings/015/briefing15.pdf
41. Xiong X, Mokbel MF, Aref WG (2005) Sea-cnn: scalable processing of continuous *k*-nearest neighbor queries in spatio-temporal databases. In: ICDE'05, Tokyo, Japan, pp 643–654
42. Yiu ML, Jensen CS, Huang X, Lu H (2008) Spacetwist: managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In: ICDE'08, Cancun, Mexico, pp 366–375
43. Yu X, Pu KQ, Koudas N (2005) Monitoring *k*-nearest neighbor queries over moving objects. In: ICDE'05, Tokyo, Japan, pp 631–642

Author Biographies

Ali Khoshgozaran received a B.S. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2003 and an M.S. degree in computer science from The George Washington University, Washington D.C., in 2005. He is currently working toward his Ph.D. degree in computer science at the University of Southern California where he is a research assistant working on geo-spatial databases, location-based services and location privacy at the Integrated Media Systems Center (IMSC).

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 U.S. Presidential Early Career Awards for Scientists and Engineers (PECASE). He is a distinguished member of ACM and a senior member of IEEE.

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