Towards Private Navigation of Tree Structured Spatial Indexes

Ali Khoshgozaran[1]
Samsung Electronics
Irvine, CA
jaffar.k@samsung.com

Cyrus Shahabi
University of Southern California, InfoLab
Los Angeles, CA 90089
shahabi@usc.edu

With many location-based services, spatial data such as points of interest are indexed at a potentially untrusted host and queries are evaluated by navigating the underlying index structure used to partition the data. While encryption can prevent the host from learning the data content (i.e., what is accessed), it cannot hide the frequency that index nodes are accessed while navigating the index for query processing. Combining the knowledge of such access frequencies with public knowledge readily available about points of interest, the host can infer sensitive information about the indexed data and hence the locations of the users querying it (violating location privacy). In this paper, we propose a technique that hides frequency access to the nodes of tree-structured spatial indexes (e.g., R-tree) from an untrusted server hosting the data. With our approach, each access to an index node requires reading an extra node using a precomputed node-based probability distribution function to guarantee uniform node access at all tree levels. We analytically verify the strong level of privacy achieved with a constant computation and acceptable communication and storage overhead for employing our private index navigation scheme.

Key Words: Location Privacy; Spatial Databases; Location-Based Services; Private Information Retrieval; Anonymity; Cloaking

1. INTRODUCTION

The ever increasing demand for querying spatial data through location-based applications has dramatically increased the popularity of location-based services (LBS). With LBS, location data such as points of interest are hosted at a location server LS and queried by users $U=\{u_1,\ldots,u_s\}$ who subscribe to the service. The location server LS (or server for short) usually employs a tree-structured spatial indexing scheme such as a kd-tree or an R-tree [5] to index data. However, LS is potentially untrusted and therefore, a common concern in LBS is achieving privacy; User queries are usually location-dependent and therefore the focus is on achieving user *location privacy*. Protecting the location information embedded in user queries as well as the data accessed to respond to user queries are the key objectives in the location privacy domain. The second goal is also referred to as *content privacy*[14].

Recently, numerous K-anonymity/spatial cloaking[6,13] and spatial transformation[7,17] techniques are proposed to achieve location privacy. Furthermore, encryption is known as the de facto solution to achieve content

[1] This work has been carried out when the author was a PhD student at USC's Information Laboratory.

privacy. However, even with encryption, severe information leakage to LS can happen by revealing how the underlying spatial index structure is accessed. Consider the following scenario. Assume N=$\{o_1,o_2,\ldots,o_n\}$ static objects (e.g., restaurants) are first encrypted and then indexed by a tree-structured index such as an R-tree R which is then stored at LS and queried by a subscriber $u \in U$. Clearly, each o_i and hence its enclosing nodes (i.e., MBRs) have a certain level of "popularity" represented by how frequently the indexed objects are requested by users. Encrypting the contents of each MBR, however, does not affect how frequently it is being accessed. Therefore, LS can associate the immediate MBR (and potentially all its ancestor MBRs) indexing the most popular object o_i with the most frequently queried node in R with high probability. Exploiting its prior knowledge of o_i popularity (e.g., the location of the hottest restaurant in Los Angeles) or how objects in N are distributed, LS can guess u's location with good enough accuracy. Protecting information about how frequently nodes are accessed is also referred to as *access privacy* [14]. It is known that without access privacy, content privacy is not fully achievable [11] and hence to protect user privacy, both content and access privacy are required at the same time.

Achieving access privacy by protecting the adversaries from learning any sensitive information from the patterns of accessing data is not trivial. The private information retrieval techniques aim to achieve access privacy by entirely blinding the server from learning any information about what records are being accessed and hence how frequently they are requested. However, PIR is very costly. Using information theoretic PIR, one can achieve perfect secrecy at the cost of a linear client server communication [3]. To reduce this prohibitively expensive cost, computational PIR schemes bring the communication cost down to $O(\sqrt{n})$ by assuming a computationally bounded server where the security of the approaches relies on the intractability of a computationally complex mathematical problem, such as Quadratic Residuosity Assumption [10]. However, similar to information-theoretic PIR, this class of approaches cannot avoid a linear scan of all database items per query. Ultimately, the class of Hardware-based PIR approaches places trust on a tamper-resistant hardware device [1]. These techniques achieve almost optimal computation and communication overhead at the cost of relying on a hardware device (with severe computing and storage resources) to provide perfect secrecy. Therefore, the PIR-based approaches to location privacy [4, 8] are still relatively costly to be employed in practice despite achieving strong user confidentiality.

In this paper we protect access privacy by proposing a novel technique with significantly less cost than the proposed PIR approaches mentioned above while still achieving strong measures of user, content and access privacy. Our scheme is based on the observation that if access privacy is achieved through protecting information leakage from the patterns of accessing index nodes, encryption and client-side query processing are enough to achieve content and user location privacy, respectively. Therefore, our contribution is proposing a secure tree navigation scheme to achieve access privacy. We flatten node access frequencies for each tree level under a practical and relaxing assumption about the variation between node access frequencies. Obviously, to achieve privacy, our method incurs extra computation, communication and storage overhead compared to processing queries on the original index. However, we analytically show the costs remain acceptable for LBS service providers and subscribers.

Our proposed technique is query-independent. In other words, it manipulates the underlying structure of the tree and abstracts away the details of tree navigation from the query processing module. Therefore, it requires minimal extra computation or storage overhead at the client side by only requiring encryption and decryption of nodes.

2. RELATED WORK

As we stated in Section 1, the bulk of existing work on privacy of spatial data focuses on achieving user location privacy. Here, the goal is to ensure a user location is protected from the untrusted server by hiding users' location information. The K-anonymity, cloaking, transformation-based and dummy-based techniques are notable examples of such approach [6, 7, 9, 13, 17]. However, these approaches fall short of protecting access privacy and hence cannot protect user location confidentiality. The server can combine the information gathered from analyzing the frequency of nodes requested during query processing with its prior knowledge about the objects indexed to deduce user locations with high probability. This approach is also referred to as the *correlation attack*. While PIR-based approaches to location privacy are resilient to correlation attacks, they incur very high computation and communication costs [4] or rely on a trusted hardware with severe computation and storage limitations [8].

Perhaps closest to our work is the techniques proposed by Lin and Candan in [11, 12]. Although the focus is on traversing XML and other structured documents rather than performing spatial queries, the proposed approaches can be applied to our set up. The authors devise schemes that use a redundancy set with a node swapping technique to obfuscate tree navigation. With both techniques, the complexity of tree traversal is $O(d \times m)$ for d and m representing the tree height and the redundancy set size, respectively. However, two fundamental differences between our approach and the techniques discussed in [11, 12] are our read-only nature of protocols and the relaxed burden on the client side. With our approach clients iteratively perform read, decrypt and request cycles with the server. In contrast, even a read request for a certain data element in [11,12] requires clients to write/modify the underlying tree structure for each node access during the index traversal. This requires the establishment of concurrency control mechanism to maintain the integrity of the tree while avoiding deadlocks. Moreover, the read/write nature of client operations during the index traversal and the locking mechanism both exacerbate client response time when server is interacting with multiple clients concurrently. Finally, while [12] assumes query load is uniformly distributed among nodes and [11] does not consider exact original node access frequencies as server's prior knowledge, we assume the server's awareness of the non-uniform query access frequencies as prior knowledge and devise schemes that prevent the server from gaining any useful knowledge from access frequencies to our proposed privacy-aware index.

Another relevant study is the recent work of Williams et al. [15] which employs Oblivious RAM to enable private retrieval of a data item. The novel reshuffling protocol proposed improves the costly computation complexity of an ORAM protocol and yields an amortized cost of $O(\log N \log\log N)$ per query. There are three key differences between this work and our proposed technique. For one, [15] places significant demands onto the client with regard to storage (an $O(\sqrt{n})$ temporary client side storage) and computation (the construction of an encrypted bloom filter, an $O(N \log\log N)$ oblivious scramble algorithm, etc.). Moreover, the proposed ORAM protocol incurs expensive communication cost; An $O(N \log\log N)$ shuffling coupled with several round trips for online query processing result in up to 100 second response time for some queries in a dataset of 100MB even under a simulated network latency setup. Lastly, the ORAM protocol is designed to retrieve a single data item. Using it in our LBS setup to process range or kNN queries requires an index traversal which results in the retrieval of multiple items further exacerbating the response time. In Section 4 we show how we achieve practical response times without placing strong demand on the client side.

3. PRELIMINARIES AND THREAT MODEL

To serve data efficiently, LS uses a tree-like index such as an R-tree R to index objects in N. Without loss of generality and to simplify discussion, we focus our attention on R-trees due to their popularity for indexing static spatial data. Since all objects are available during the offline index (i.e., R) construction, we also assume all tree nodes are filled with data. We relax both of these assumptions in Section 4.3 and show how similar techniques can be applied to R-trees with partially full nodes and to other tree-structured spatial indexes such as kd-trees and quadtrees.

Table I. Notations and symbols

Symbol	Meaning
N	POI data
R, R'	R-tree, Privacy-aware R-tree
$N_{i,j}(N'_{i,j})$	j^{th} node of $R(R')$ at depth i
c	node capacity
H	one-way hash
RS	query result set
$m_1 \dots m_c$	$N_{i,j}$'s children each an MBR
$f_{i,j}(f'_{i,j})$	normalized access frequency of $N_{i,j}(N'_{i,j})$
$N_{i,j}.m_k \rightarrow N_{i,j'}$	assigning m_k to $N_{i,j'}$
$r_{i,k \rightarrow i,k'}$	probability of including $N_{i,k'}$ in $N_{i,k}$'s RS
S_i	sequence of siblings at depth i

Assuming R's capacity is c, each node contains c children represented by $m_1,\dots,m_c$ where each m_i is itself an MBR of c objects (by using the term "objects" hereafter, we refer to internal elements of each node which could be MBR's of lower nodes or the actual POI data for leaf nodes. To avoid confusion, we explicitly use "leaf objects" to refer to the latter case).We use the notation $N_{i,j}$ to refer to the j^{th} node of R at depth i={1,…,h}. Figure 1 illustrates such a tree where leaf nodes are represented by $N_{h,j}$ indexing $o_1,\ldots,o_n$ represented by their location and identifiers. Table 1 summarizes the notations used throughout the paper.

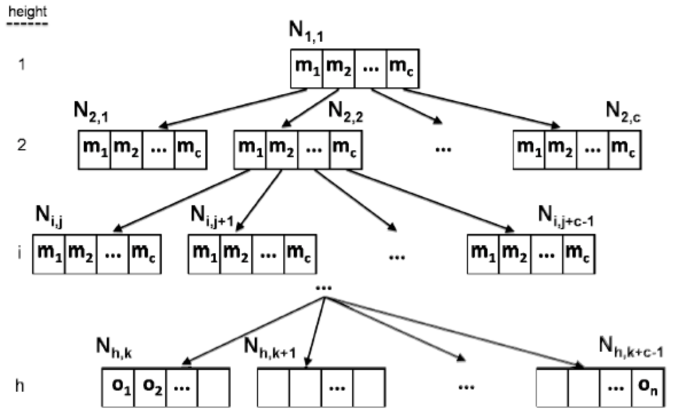


Figure 1. R-tree

Using its prior knowledge of past queries, LS which hosts R, knows how frequently internal and leaf nodes of R are requested. We use $f(N_{i,j})$, or its short form $f_{i,j}$, to show the normalized ($0 \le f_{i,j} \le 1$) access frequency of the node $N_{i,j}$ in T number of queries. We also assume the strongest adversarial case where LS is aware of the spatial distribution of elements in N. To achieve privacy, we replace R with its privacy-aware variant R' and have LS serve queries using R' whose nodes and their access frequencies are denoted by $N'_{i,j}$ and $f'_{i,j}$, respectively. We assume the trend in which objects are queried (i.e., their popularity) does not vary significantly with time and in particular after replacing R with R'. Each $N'_{i,j}$ in R' is assigned a node identifier id = H(i,j) computed as a one-way hash function of i, j (e.g., SHA512) and is encrypted with a private key inaccessible to LS to protect content privacy. Similar to [11,12,16] we assume the secret key is shared by subscribers of LS to decrypt R-tree nodes. To avoid collusion attacks, the cryptographic operations can be performed by the assistance of inexpensive smartcards placed in subscriber's client devices [2]. We also presume users employ an anonymous access protocol to protect their identity while interacting with LS.

Since nodes are encrypted, LS cannot traverse R' and tree navigation becomes an interactive scheme between the user u and the server. To perform any spatial query such as range or kNN, u privately requests a series of nodes $N'_{i,j}$ chosen based on the query processing logic. At each step u first requests the next node $N'_{i,j}$ by its id= H(i,j). After decrypting $N'_{i,j}$, u identifies one of the node's children $N'_{i+1,j'}$ for further expansion and sends a subsequent request to the server for the node represented by id = H(i+1,j'). This read, decrypt, encrypt and request process is repeated by u until the leaf nodes (likely) containing the query result are retrieved. If several MBRs at level i intersect with a query, they are each retrieved separately using the scheme discussed above. Note that encrypting each MBR in a node as opposed to encrypting the whole node data would result in an information leakage by exposing the ordering among the elements of each node.

Since both R and R' are hosted by the location server LS, it knows $f_{i,j}$ values while serving R and later learns $f'_{i,j}$ values by serving R'. The location server is not trustworthy and is curious to exploit this knowledge to infer user locations from R and R' node access frequencies. To do this, LS employs a *frequency variation attack*. That is, the server tries to correlate the $f_{i,j}$ and $f'_{i,j}$ values or exploits the variations among $f'_{i,j}$ values in R' and combines this with its prior knowledge to infer the contents of $N'_{i,j}$. Our approach presented in Section 4 shows how we generate a privacy-aware tree R' whose $f'_{i,j}$ values are meaningless to LS thus achieving user location privacy.

4. OBFUSCATING ACCESS FREQUENCIES

Consider the scenario where LS hosts N and the client (we hereafter use the terms client and user interchangeably) u forms a query Q to find a nearby object o_i. As we discussed in Section 1, one extreme solution (in terms of both privacy and efficiency) is for u to use PIR to privately navigate the index structure hosted by LS to prevent him from learning her location. While remaining perfectly secret, this approach is very expensive (Section 2). In another extreme, K-anonymity can be used to confuse LS by sending him K queries to make u indistinguishable among a redundancy set of size K. Although being efficient, this technique offers significantly weaker privacy guarantees as in the best case, it makes u indistinguishable among a usually small set of K-1 other users. Moreover, recent studies [4,7,8,17] show how sophisticated attacks can be mounted against such schemes. We strike a compromise by utilizing the hierarchical nature of tree structured spatial indexes to enable efficient yet oblivious traversal of the tree by making node access frequencies meaningless to the untrusted server. With our approach, the original R-tree R is replaced by its privacy aware variant R' such that the index navigation is performed on R' nodes denoted by $N'_{i,j}$. To process Q, u interactively requests a series of nodes from LS that allow her to find o_i. We show that with our approach, the server does not learn the frequency (and hence content) of nodes accessed to execute Q.

The basic idea behind our proposed scheme is to perform redundant reads from the server for each node requested by u. We obfuscate node access frequencies by grouping “less popular” nodes with more frequently accessed ones in a redundancy set RS. We compute for each node $N'_{i,j}$, a probability distribution function stored at its parent node. This function instructs the client to form a redundancy set RS which includes $N'_{i,j}$ as well as another node of the same height chosen according to a certain precomputed probability function to make node access entirely uniform. More formally, our goal is to break the correlation between the original node access frequencies $\{f_{i,1},f_{i,2},\ldots,f_{i,c^{i-1}}\}$ and the modified access frequencies of R' nodes $\{f'_{i,1},f'_{i,2},\ldots,f'_{i,c^{i-1}}\}$ for $i \in \{1,\ldots,h\}$ by making node access frequencies uniform.

To achieve uniformity in node access frequencies, one can think of creating a redundancy set RS for each $N'_{i,j}$ access in R' with capacity c. In particular, each RS would include the original node requested as well as K other elements chosen uniformly at random from the c^{i-1} nodes with the same height to protect node access frequencies. In the following lemma, we show that this naive protocol does not protect information leakage because the resulting node access frequencies $f'_{i,j}$ are not uniform and are in fact highly correlated with $f_{i,j}$ values. More formally, we show that the histogram of access frequencies would uniformly increase for all nodes and thus the new frequency histogram will not be uniform since the original frequencies are not uniform.

Lemma 1. Using the above naïve scheme, $f'_{i,j}=\alpha+\beta\times f_{i,j}$ where $a = p_i \sum_{1\le k\le c^{i-1}} f_{i,k}$, $\beta = 1 - \frac{1}{p_i}$ for $p_i = \frac{K}{c^{i-1}-1}$.

Proof. By randomly choosing elements of RS, at depth i, each $N_{i,k}$, $k\neq j$ has a $p_i = \frac{K}{c^{i-1}-1}$ chance of being included in RS. Such added nodes each contribute to $f'_{i,j}$ with probability p_i. In other words, including $N_{i,k}$ in $N_{i,j}$'s redundancy set adds $f_{i,k}$ to $f'_{i,j}$ with probability p_i. In general,

$$f'_{i,j} = f_{i,j} + p_i \sum_{k\neq j} f_{i,k} \tag{1}$$

$$f'_{i,j} = f_{i,j} + p_i \left(\sum_{1\le k\le c^{i-1}} f_{i,k} - f_{i,j} \right) \tag{2}$$

$$f'_{i,j} = \frac{p_i - 1}{p_i} f_{i,j} + p_i \sum_{1\le k\le c^{i-1}} f_{i,k} \tag{3}$$

$$f'_{i,j} = \alpha + \beta f_{i,j} \tag{4}$$

Corollary: For $N'_{i,j1}$, $N'_{i,j2}$: $f'_{i,j1}-f'_{i,j2}= \beta\times(f_{i,j1}- f_{i,j2})$.
The above observation states that members of the redundancy set cannot be uniformly picked if the original nodes are accessed at different frequencies. Therefore, we need to take values of $f_{i,j}$ into account while constructing the redundancy set. We now present a protocol that achieves this goal.

4.1 Probabilistic Uniform Node Access

Consider the tree of Figure 2 where each $N_{i-1,j}$ access results in a subsequent request for one of the nodes in the next level of the tree (i.e., i) at different frequencies. Our goal is to add one redundant node to each original node request in such a way that the overall node access frequencies $f'_{i,j}$ for all nodes at depth i become equal (hereafter, we refer to this criteria as the uniform node access frequency). To achieve this, we define for each internal node $N_{i-1,j}$ a probability table $pt_{i-1,j}$ whose values are of the form $r_{i,j\rightarrow i,k}$ denoting the probability of accessing each $N_{i,k}$

whenever $N_{i,j}$ is originally requested. In other words, for each $N_{i,j}$ request at level i-1, one $N_{i,k}$ is added to a redundancy set RS in a random order according to $r_{i,j \rightarrow i,k}$ values in $pt_{i-1,j}$. In order to enforce one and only one redundant read per node access, we require that $\sum_{k \in S_i - j} r_{i,j \rightarrow i,k} = 1$ which means each node $N_{i,j}$ is paired with one $N_{i,k}$ as its redundant read with probability $r_{i,j \rightarrow i,k}$. Figure 3 illustrates how a redundant node is chosen. Each node picks a random variable $x \in (0,1]$ and drops it the large rectangle whose length is 1. The small rectangle containing x identifies the redundant node to be read. Note that even for a fixed node $N_{i,j}$ redundant nodes can be different each time as the value of x is determined randomly for each $N_{i,j}$ request.

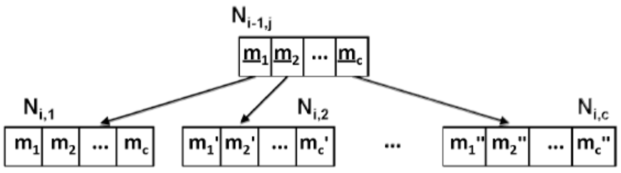
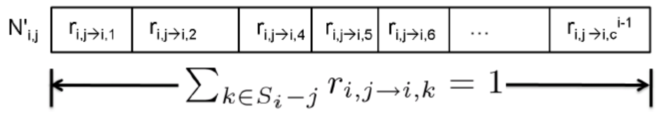


Figure 2. Original R-tree

To store each $pt_{i-1,j}$, for an internal node we form a $c \times S_i$ matrix where $S_i = \{1,\ldots,c^{i-1}\}$ and each (encrypted) entry represents $r_{i,j \rightarrow i,k}$. In order to maintain the original capacity (fan-out) of the tree structure, we store these tables in a separate data structure where each $pt_{i-1,j}$ is identified by $N_{i-1,j}$'s id. This probability table result in a storage overhead to prevent leakage from the pattern the index is traversed. With our implementation, maintaining these tables incurred a reasonable increase in total amount of space needed to store the index for different values of c.


$N'_{i,j}$
$r_{i,j \rightarrow i,1}$
$r_{i,j \rightarrow i,2}$
$r_{i,j \rightarrow i,4}$
$r_{i,j \rightarrow i,5}$
$r_{i,j \rightarrow i,6}$
...
$r_{i,j \rightarrow i,c^{i-1}}$
$\sum_{k \in S_i - j} r_{i,j \rightarrow i,k} = 1$
Figure 3. Probability contributions of node $N'_{i,3}$

We proceed to the offline calculation of the probability tables. Since the size of the redundancy set RS is always 2, using the above scheme each node access adds to the access frequency of one and exactly one other node included in its RS. To compute the modified node access frequency $f'_{i,j}$ we add to its original access frequency $f_{i,j}$, the weighted probabilistic frequency contributions of all other nodes of depth i. This contribution is a function of each node's popularity, as well as the probability of that node picking $N_{i,j}$ as its redundant node. More formally, performing the above scheme increases $f_{i,j}$ to $f'_{i,j}$ for $j \in \{1,\ldots,c^{i-1}\}$:

Lemma 2. The new access frequency of a node $N_{i,j}$ will increase by the sum of all other node's probabilistic contribution to $N_{i,j}$ at depth i. Or $f'_{i,j} = f_{i,j} + \sum_{k \neq j} f_{i,k} \times r_{i,k \rightarrow i,j}$. Using lemma 2, we prove the following property.

Theorem 1. The above scheme increases the sum of access frequencies at each depth by a factor of 1. Or $\sum_{j \in S_i} f'_{i,j} = 2 \sum_{j \in S_i} f_{i,j}$.

Proof: Using lemma 2 and setting $\sum_{k \in S_i - j} r_{i,j \to i,k} = 1$, we write:

$$\sum_{j \in S_i} f'_{i,j} = \sum_{j \in S_i} f_{i,j} + \sum_{j \in S_i} \left\{ f_{i,j} \times \sum_{k \in S_i - j} r_{i,j \to i,k} \right\} \tag{5}$$

$$\sum_{j \in S_i} f'_{i,j} = \sum_{j \in S_i} f_{i,j} + \sum_{j \in S_i} \{ f_{i,j} \times 1 \} = 2 \sum_{j \in S_i} f_{i,j} \tag{6}$$

To clarify, let us take the following examples for the simple case of a tree with height h = 2. For c = 2 (Figure 4a), this scheme is straightforward. We have $f'_{2,1} = f_{2,1} + f_{2,2} \times r_{2,2 \to 2,1} = f_{2,1} + f_{2,2} = 1$. Similarly, $f'_{2,2} = 1$. In other words, each request for one node includes the other with probability 1. Therefore, the probability of accessing both nodes is equal. For higher values of c such as c=3, the case is slightly more complicated (see Figure 4b). To obtain new node access frequencies, we need to solve the following system of equations.

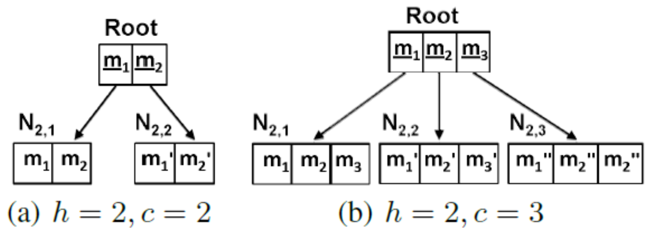


Figure 4. Examples

$$\left\{ \begin{array}{l} f'_{2,1} = f_{2,1} + f_{2,2} \times r_{2,2 \to 2,1} + f_{2,3} \times r_{2,3 \to 2,1} \\ f'_{2,2} = f_{2,2} + f_{2,1} \times r_{2,1 \to 2,2} + f_{2,3} \times r_{2,3 \to 2,2} \\ f'_{2,3} = f_{2,3} + f_{2,1} \times r_{2,1 \to 2,3} + f_{2,2} \times r_{2,2 \to 2,3} \\ r_{2,1 \to 2,2} + r_{2,1 \to 2,3} = 1 \\ r_{2,2 \to 2,1} + r_{2,2 \to 2,3} = 1 \\ r_{2,3 \to 2,2} + r_{2,3 \to 2,1} = 1 \end{array} \right\} \tag{7}$$

The above system has 9 unknowns (i.e., 3 new f' and 6 new $r_{i,j \to i,k}$ unknowns for $k \in S_i$-j) and 6 equations. However, a closer look at the protocol gives us the remaining information required to solve the above system. Based on our objective of equalizing modified node access frequencies, we have $f'_{2,1} = f'_{2,2} = f'_{2,3}$. Using Theorem 1, $f'_{2,1} + f'_{2,2} + f'_{2,3} = 2$. This property gives us three less unknowns:

$$\left\{ \begin{array}{l} f'_{2,1} = \frac{2}{3} \\ f'_{2,2} = \frac{2}{3} \\ f'_{2,3} = \frac{2}{3} \end{array} \right\} \tag{8}$$

Therefore, for the general case, in order to achieve uniform access for each tree depth i we set $f'_{i,j} = \frac{2\sum_{j \in S_i} f_{i,j}}{|S_i|}$ (observe that $|S_i|$ is equal to the number of nodes at level i) and solve a linear system of e equations and e unknowns offline. The unknowns are $|S_i|$-1 probability contributions for each of the $|S_i|$ nodes. Thus, e= $|S_i|\times(|S_i|-1)= |S_i|^2-|S_i|$.

To execute a query Q, at each step the user u receives the encrypted original ($N'_{i,j}$) and redundant ($N'_{i,j'}$) MBRs along with their probability tables. Next, u discards $N'_{i,j'}$ and its table (which needs to be transferred to u to prevent LS from identifying the redundant MBR), decrypts $N'_{i,j}$ and picks the next MBR from level i+1 of $N'_{i,j}$ to be expanded and uses the probability table to pick the redundant MBR for the next original node request. This process is repeated for every node u requests as part of processing Q. Note that even for the first query at time T= t_0, the likelihood of any two nodes being included in the user request is $\binom{|S_i|}{2}$. Finally, aside from the storage overhead, the security of this method comes at the cost of transferring two probability tables for each node request to the client.

The probability contributions of each node (derived from solving the system of equations for each probability table) determine which redundant node will accompany each originally requested node in client's request to the server. However, one might wonder if the actual node access frequencies will need time to converge to our calculated values after certain number of node requests at T=t_1, t_1>>t_0. This is in fact not the case. Consider the example of Figure 4b and let $f_{2,1}$=0.6, $f_{2,2}$=0.1 and $f_{2,3}$=0.3. Solving a system of 6 equations and unknowns will yield

$$\begin{Bmatrix} r_{2,1\to2,2} = 0.54 \\ r_{2,1\to2,3} = 0.45 \\ r_{2,2\to2,1} = 0.07 \\ r_{2,2\to2,3} = 0.93 \\ r_{2,3\to2,1} = 0.20 \\ r_{2,3\to2,2} = 0.80 \end{Bmatrix} \tag{9}$$

For the first query submitted at T=t_0, we calculate $p(N_{2,1},N_{2,2})$ which represents the probability of nodes $N_{2,1}$ and $N_{2,2}$ belonging to the user's request.

$$p(N_{2,1}, N_{2,2}) = f_{2,1} \times r_{2,1\to2,2} + f_{2,2} \times r_{2,2\to2,1} = 0.6 \times 0.54 + 0.1 \times 0.07 = \tfrac{1}{3} = \tbinom{3}{2}$$
$$p(N_{2,1}, N_{2,3}) = p(N_{2,2}, N_{2,3}) = \tfrac{1}{3} = \tbinom{3}{2}$$

Therefore, any two nodes could be the first nodes requested from the server. To see why observe that in Equation 8, we set similar values for the expected node access frequency $f'_{i,j}$. Therefore, each node is equally likely to be present in a user's request. It is important to note that although node access frequencies are equal, one cannot take away the attacker's prior knowledge of objects popularity. For instance, consider an extreme case where the client downloads the entire database and processes her query locally to achieve perfect secrecy. Even though all nodes are accessed once (and in fact transferred to the client), the server can still assign a higher probability to a certain object being the actual user intended object. However, this knowledge is not acquired from monitoring the client/server interactions. Similarly, our goal is to ensure our methods do not leak any extra information to the attacker by the way objects are accessed.

4.2 Security and Complexity Analysis

In this section we review the server overhead in employing our proposed probabilistic uniform node access scheme, henceforth denoted by PU, for spatial query processing using the privacy-aware tree R'.

Theorem 2. Let $C_O(n)$, $C_{PU}(n)$ denote the computational cost of processing a spatial query Q over n objects using the original and the PU method, respectively. $C_{PU}(n) \approx 2C_O(n)$.

Proof: Processing Q using R at each step, requires a "visit" phase to inspect a node $N_{i,j}$ and an "expand" phase where the server identifies the next MBR to be visited. This MBR is selected based on the nature of Q (for instance an MBR overlapping with Q if Q is a range query). Replacing R with R' results in two changes. First, the MBR inspection process is shifted to the client u as nodes of R' are encrypted. Furthermore, each "visit" to a single node is replaced by requesting a redundancy set of size υ where $\upsilon = 2$ for PU. However, the expansion phase remains intact because of all υ nodes requested by u, only one MBR would trigger the next client server interaction (see Figure 5). Therefore, the complexity of processing Q is $\upsilon C_O(n)$ where υ is a constant. The ≈ notation is used to account for the (constant) extra cost of retrieving the probability tables from the server.

Therefore, the PU method achieves fully uniform node access frequency where the probability of any two nodes being requested is $\binom{|S_i|}{2}$. Moreover, according to Lemma 2, the server overhead for the PU technique is twofold due to an increase in size of the result set.

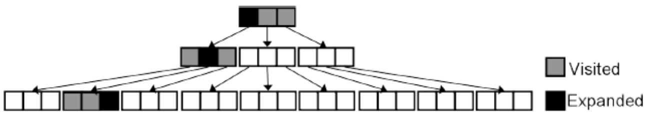


Figure 5. Visiting vs. expanding nodes

4.3 Generalizations

We assumed in Section 3 that leaf objects in N are indexed using an R-tree with capacity c where each node contains exactly c elements and deferred the generalization of our approach to partially full R-trees as well as applying our proposed scheme to oblivious navigation of other tree-structured spatial indexes such as kd-trees and quadtrees. In the following sections, we discuss these two generalizations, respectively.

Partially Full R-trees: To prove several properties of our proposed R-tree variant, we assumed each internal node $N'_{i,j}$ of R-tree includes exactly c child MBRs and each leaf node groups exactly c objects into a leaf MBR. Although having the entire dataset available offline enables the construction of a balanced tree where most of the nodes are in fact full, several nodes at each level might not contain exactly c objects. The partially full structure of the tree, even though nodes are encrypted, can potentially leak information to the server about the distribution of the actual static objects. To deal with this issue, during an offline process, we traverse the tree from level h-1 up to level 1 and for each parent node $N'_{i,j}$ with c'<c children, we pad c'-c children with dummy data and access frequency of zero and link them to $N'_{i,j}$. This process guarantees that no information is leaked to the server from any asymmetry of the tree structure. Moreover, one can easily verify that using the above approach, all tree properties and proofs of correctness and security still hold. Finally, observe that all proofs of complexity assumed a worst case scenario where the tree nodes are all full and therefore, padding the nodes with dummy data does not exacerbate the complexity analysis of earlier sections.

Other Tree Structured Spatial Indexes: In previous sections, we detailed our schemes for privacy-aware navigation of R-trees. Although we focused our attention on R-trees, we did not make any assumptions specific to R-trees that do not hold in other tree structured spatial indexes. The PU technique discussed in Section 4.1 can be employed with any other tree-structured index that respects the notion of recursively grouping lower level objects in higher nodes. Furthermore, the tree should have the same number of objects in each node. Obviously, several spatial indexes such as kd-trees and quadtrees satisfy all the above properties. Finally, since tree nodes are encrypted, the client should be capable of performing the query processing interactively with the server. This requires the client to be aware of how the underlying spatial index is used for query processing.

4.4 Limitations of Probabilistic Uniform Node Access Method

Although the system of equations derived from Theorem 1 can be easily solved using Gaussian reduction, there are cases where the solution is not valid. To see why note that

$$f'_{i,j} = \frac{2\sum_{j \in S_i} f_{i,j}}{|S_i|} = f_{i,j} + \sum_{k \neq j} f_{i,k} \times r_{i,k \to i,j}$$

If $\{\exists i, j \ s.t. \ f_{i,j} \geq \frac{2\sum_{j \in S_i} f_{i,j}}{|S_i|}\}$, there will at least exist one value of $r_{i,k \to i,j}$ smaller than 0 for some k which is an invalid probabilistic contribution for a node. This situation occurs if there are large variations among the access frequencies of nodes of a certain tree level. For instance, if c = 3, an optimal solution can be found only if $\forall i, j : f_{i,j} \leq 2/3$. One way to solve this problem is to replicate a node whose $f_{i,j}$ is more than the above threshold into two identical nodes each with half the original frequency. To maintain the structure of the tree, it suffices to randomly choose one of the replicated nodes at its parent with probability 1/2.

However, replication introduces new complexities to our approach. Therefore, this approach is effective whenever there is normal variations among $f_{i,j}$ values at each depth. We believe this is a practical assumption for POI data. However, applying this technique to cases where this property does not hold is an open question of our approach. In the remainder of this paper, we discuss our future work to go beyond the limitations of probabilistic node access technique and devise techniques that do not assume restrictions on node access frequency variations and work well even in the presence of outliers.

4. CONCLUSION AND FUTURE WORK

To protect the location privacy of users who subscribe to location-based services, encryption is not sufficient for hiding the contents of the underlying spatial index. The potentially untrusted server hosting spatial data can obtain sensitive information by monitoring "how" (i.e., in what frequency) the encrypted index nodes are being accessed. Combining this information with its prior knowledge about the data, the server can easily deduce the location of the user querying the data. In this paper, we proposed our probabilistic uniform node access technique that enables oblivious navigation of tree-structured spatial indexes while incurring acceptable communication overhead and server side complexity and imposing minimal burden on the client side. We analytically studied the security and efficiency of our approach. As we discussed in Section 4.4, our proposed technique imposes some assumptions on the distribution of access frequency values for nodes. Moreover, it requires storing probability values at tree nodes. To address these two issues, we are working on an object permutation scheme that obfuscates the histogram of node access frequencies in the original tree by converting it to a semi-uniform distribution. Our initial observations demonstrate significant savings on computation and communication overhead compared to our approach we proposed in this paper. However, such benefits obviously come at a cost which in our case will be more lax the privacy guarantees. We plan to empirically evaluate both approaches with real-world and synthetic data to better understand their properties.

Finally, while efficient for querying static data, our original and modified index structures are not suitable for processing dynamic data. As part of our future work, we are investigating how to improve our proposed methods to efficiently deal with dynamic data.

ACKNOWLEDGEMENTS

This research has been funded in part by NSF grant CNS- 0831505 (CyberTrust), the NSF Integrated Media Systems Center (IMSC) and unrestricted cash and equipment gift from Google and Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] D. Asonov. Querying Databases Privately: A New Approach to Private Information Retrieval, volume 3128 of Lecture Notes in Computer Science. Springer, 2004.

[2] L. Bouganim and P. Pucheral. Chip-secured data access: confidential data on untrusted servers. In VLDB'02, pages 131–142.

[3] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. J. ACM, 45(6):965–981, 1998.

[4] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In SIGMOD'08, pages 121–132.

[5] A. Guttman. R-trees: a dynamic index structure for spatial searching. In SIGMOD'84, pages 47–57.

[6] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving anonymity in location based

services. A Technical Report, 2006.

[7] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In SSTD'07, pages 239–257.

[8] Ali Khoshgozaran, Cyrus Shahabi, and Houtan Shirani-Mehr, Location privacy: going beyond K-anonymity, cloaking and anonymizers, Knowledge and Information Systems, Volume 26, Issue 3 (2011), Page 435.

[9] H. Kido, Y. Yanagisawa, and T. Satoh. Protection of location privacy using dummies for location based services. In ICDE Workshops, page 1248, 2005.

[10] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally private information retrieval. In FOCS'97, pages 364–373.

[11] P. Lin and K. S. Candan. Secure and privacy preserving outsourcing of tree structured data. In Secure Data Management, VLDB Workshop, SDM, pages 1–17, 2004.

[12] P. Lin and K. S. Candan. Hiding tree structured data and queries from untrusted data stores. Information Systems Security, 14(4):10–26, 2005.

[13] M. F. Mokbel, C.-Y. Chow, andW. G. Aref. The new casper: Query processing for location services without compromising privacy. In VLDB'06, pages 763–774.

[14] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. IBM Syst. J., 40(3):683–695, 2001.

[15] P.Williams, R. Sion, and B. Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In CCS'08, pages 139–148, 2008.

[16] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis. Outsourcing search services on private spatial data. In ICDE'09, pages 1140–1143.

[17] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In ICDE'08, pages 366–375.