OST-Tree: An Access Method for Obfuscating Spatio-Temporal Data in Location Based Services

Quoc Cuong TO, Tran Khanh DANG*
Faculty of Computer Science & Engineering, HCMUT
Ho Chi Minh City, Vietnam
{qcuong, khanh}@cse.hcmut.edu.vn

Josef KÜNG
FAW Institute, Johannes Kepler University
Linz, Austria
josef.kueng@faw.jku.at

Abstract—Since the development of location-based services, privacy-preserving has gained special attention and many algorithms aiming at protecting user’s privacy have been created such as obfuscation or k-anonymity. However, all of these researches separate the algorithms from the database level. Thus, the querying process has two phases, querying the database to retrieve the accurate positions of users and then modifying them to decrease the quality of location information. This two-phase process is time-consuming due to the number of disk accesses required to retrieve the user’s exact position. We address this problem by proposing OST-tree, a structure that embeds the user’s privacy policy in its node and obfuscates the spatio-temporal data. Experiments show that OST-tree provides an improvement over the algorithm separated from the database level for both querying costs and user’s privacy protection.

Keywords-index; spatio-temporal; obfuscation; LBS; privacy

I. INTRODUCTION

With the rapid development of global positioning system (GPS), there are more than 4.5 billion mobile users by the year 2009 and the number is expected to increase more. Among the various services for mobile phone, the location-based service (LBS) is the most promising one since it supplies users with many value-added services. In order to benefit from these services, users, however, have to reveal their sensitive information such as their current locations. Such novel services pose many challenges because users are not willing to reveal their sensitive information but still want to benefit from these useful services.

To solve this privacy-preserving problem, a variety of algorithms are suggested to hide personal information of users but still allow them to use services with acceptable quality. The general idea of these algorithms is to obfuscate the user’s position [3,4,16], or to anonymize location information [1,2]. There are two limitations of these algorithms. First, all of them deal with only spatial obfuscation, but not temporal one. Second, these algorithms are separated from the database level. This separation makes the algorithms go through two phases: retrieving the exact location of user on the database level first, and then obfuscating this information on the algorithm level. This two-phase process is time-consuming due to the number of disk access required to retrieve user’s exact position.

Motivated by this, in this work, we propose a new spatio-temporal index structure based on TPR-tree [5] that can contain privacy-preserving information on it. This index structure deals with not only spatial but also temporal obfuscation, so we term it OST-tree, where OST stands for Obfuscating Spatio-Temporal data. Furthermore, because this index structure embeds privacy information on its node, the process of calculating the obfuscated data can be done in only one phase: traversing the index structure to retrieve the appropriately obfuscated data. This one-phase process can reduce the processing time considerably comparing to the two-phase process mentioned above.

In location-based services environment, each user uses many services from various service providers. For each service provider, user just wants to reveal his location in a specific degree of accuracy depending on the user’s trust on service providers. For example, users are willing to reveal their exact location to emergency services, but only reveal their obfuscated locations to advertising services. Therefore, classification of location-based service providers according to user’s trust is very necessary in order not to violate privacy policy of users. Towards this goal, OST-tree is designed to feature service provider classification by letting users specify authorizations and put the privacy information on the tree nodes.

The crucial contributions of this paper are to define the new concept of temporal obfuscation, and to embed user’s privacy policy into TPR-tree. This new index is capable of obfuscating spatio-temporal data and provides an improvement over the algorithm separated from the database level for both querying costs and user’s privacy protection.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces the new concept of temporal obfuscation, and the authorization model. Section 4 presents our OST-tree, privacy overlaying, query processing algorithms and related analyses. Section 5 shows experimental results. Finally, section 6 concludes the paper.

II. RELATED WORK

A. Location Obfuscation

Among the most popular techniques to protect user’s location, obfuscation gains much interest [10,11,16]. Location obfuscation aims at hiding user’s exact location by decreasing the quality of user’s location information. In [3], obfuscation techniques are proposed by enlarging the area containing user’s exact position. The bigger the area, the harder an attacker can infer the user’s exact location. If the area, however, is too big,
it can affect the quality of location-based services. So, it is the responsibility of user to decide which degree of accuracy of user’s location to be revealed to which service providers.

Motivated by this, Dang et al. developed the general architecture [7] to classify LBS service providers depending on the user’s trust. This architecture inherits the property of mandatory access control to label service providers so that users only reveal their locations on an appropriate level based on the labels assigned to service providers. However, the index structure in this architecture does concern about temporal data at a very abstract level. Thus, it is necessary to concretize this structure by a suitable spatio-temporal index and this will be discussed in the next section.

B. Spatio-Temporal Structures for Indexing the Present and Future Positions of Moving Objects

Several recent research focuses on indexing the present and future positions of moving objects [12] and the most popular category is parametric spatial access. Two popular access methods in this category are PR-tree and TPR-tree. PR-tree [14], however, is only suitable for objects with spatial extent. So, in applications concerning a user’s position which is a spatial point in nature, the PR-tree is not the best solution. For TPR-tree [5], it inherits the idea of parametric bounding rectangles in R-tree [15] to create time-parameterized bounding rectangle (tpbr). Since the tpbr is organized in hierarchical form in terms of space, TPR-tree is chosen as the base structure of our proposed structure so that we can easily overlay the obfuscated data in TPR-tree’s node hierarchically.

In TPR-tree, the position of an moving object \( x(t) \) at a future time \( t \ (t \geq t_0) \) is found by applying the linear function representing its location to the current time \( x(t) = x(t_0) + v(t - t_0) \) where \( t_0 \) is the initial time, \( t \) the current time, \( x(t_0) \) the initial position and \( v \) the velocity. The tpbr is also a function of time. Specifically, the lower (upper) bound of a tpbr is set to move at a real timestamp \( t_0 \) with the probability:

\[
P(t_0 \in \{\ell, \ell\}) = 1
\]

Definition 1. (Temporal obfuscation) The obfuscated value of timestamp \( t_0 \) is the temporal interval \([\ell, \ell]\) which includes the real timestamp \( t_0 \) with the probability:

\[
P(t_0 \in \{\ell, \ell\}) = 1
\]

Definition 2. (Spatio-temporal obfuscation) The obfuscated value of user’s exact position \((x_0, y_0)\) at a timestamp \( t_0 \) is a rectangular area \((x_c, y_c, w, h)\) centered on the geographical coordinates \((x_c, y_c)\) with width \( w \), height \( h \), at a temporal interval \([\ell, \ell]\), which includes the user’s exact position \((x_0, y_0)\) at a real timestamp \( t_0 \) with the probability:

\[
P((x_0, y_0) \in \text{Rectangle}(x_c, y_c, w, h)) \text{ AND } t_0 \in \{\ell, \ell\} = 1
\]

In [13], a unified index for location and profile data is proposed. This index clusters the customers based on their profiles using a categorical clustering algorithm, and then constructs a TPR-tree for each cluster. A query is processed in the profile database to retrieve the target clusters and then traverse these clusters to retrieve the customers who satisfy the criteria. This unified index is, however, used for marketing purpose which retrieves the group of interested customers, but does not concern about obfuscating the customer’s location.

It is evident from the above discussion that currently there does not exist any spatio-temporal index structure that can effectively handle spatio-temporal obfuscation. Towards this goal, in this paper, we propose the OST-tree, a structure originally motivated by the TPR-tree, but with several modifications to support spatio-temporal obfuscation.

III. TEMPORAL OBfuscATION

Many of the research activities have been done in the area of spatial obfuscation [3,4,10,11,16], but, to the best of our knowledge, no mature proposals for obfuscating the temporal data of users exist. So, we focus on this issue in this section.

Similar to spatial obfuscation, temporal obfuscation will degrade the exact value of time \( t_0 \) to the vague temporal value \([\ell, \ell]\), where \( \ell < t_0 < \ell \). For example, instead of saying that “the position of user will be in location \((x_0, y_0)\) in the next 15 minutes”, we can obfuscate the time value by saying that “the position of user will be in location \((x_0, y_0)\) in the next 13 to 16 minutes”. By combining the spatial and temporal dimension, a spatio-temporal value can be calculated by obfuscating both the spatial and temporal value. For example, according to the above example, we can say: “The user’s position is somewhere in the area of 1.2 square kilometer, including the location \((x_0, y_0)\), and within the next 13 to 16 minutes in the future”.

Definition 1. (Temporal obfuscation) The obfuscated value of timestamp \( t_0 \) is the temporal interval \([\ell, \ell]\) which includes the real timestamp \( t_0 \) with the probability:

\[
P(t_0 \in \{\ell, \ell\}) = 1
\]

Definition 2. (Spatio-temporal obfuscation) The obfuscated value of user’s exact position \((x_0, y_0)\) at a timestamp \( t_0 \) is a rectangular area \((x_c, y_c, w, h)\) centered on the geographical coordinates \((x_c, y_c)\) with width \( w \), height \( h \), at a temporal interval \([\ell, \ell]\), which includes the user’s exact position \((x_0, y_0)\) at a real timestamp \( t_0 \) with the probability:

\[
P((x_0, y_0) \in \text{Rectangle}(x_c, y_c, w, h)) \text{ AND } t_0 \in \{\ell, \ell\} = 1
\]

In our work, we have the same assumption as in [10] which states that the probability distribution of user’s position within an area is uniform. Formally, the joint probability density function \( f(x, y) \) of a region is:

\[
f(x, y) = \begin{cases} 
1 & \text{if } (x, y) \in r \\
0 & \text{otherwise}
\end{cases}
\]

where \( s(r) \) represents the area of \( r \).
Similarly, the probability distribution of user’s position within an area \( r \) and at a time \( t_0 \) within an interval \([t', t']\) is:

\[
f_{\alpha}(x, y) = \begin{cases} 
\frac{1}{s(r')} \frac{1}{t'} & \text{if } (x, y) \in r, t_0 \in [t', t'] \\
0 & \text{otherwise}
\end{cases}
\]

(4)

Definition 3. (Authorization) An authorization \( \alpha \) is a 4-tuple \(<id_{sp}, id_{user}, \Delta s, \Delta t>\) where \( id_{sp} \) is the identity of service provider, \( id_{user} \) is the identity of user, \( \Delta s \) is the degree of accuracy of user’s position (spatial data) and time, respectively. The meaning of an authorization is that a user with the identity \( id_{user} \) allows only the service provider with the identity \( id_{sp} \) to access his/her sensitive information of position and time with the degree of accuracy of \( \Delta s, \Delta t \), respectively.

For example, a user with the identity \#U232 is willing to reveal his position in the next 10 minutes with the accuracy of position and time being 600 square meters, 3 minutes, respectively, to the advertising service with the identity \#S101. This authorization can be expressed as \( \alpha_1 = <#S101, #U232, 600m^2, 3m> \). If the user’s exact position in the next 10 minutes is located at a coordinate \(<x_0, y_0>\), the result returned from the next 9 to 12 minutes to the service provider is a rectangle which has the area of 600 square meters and contains the coordinate \(<x_0, y_0>\) in case of time and position, respectively.

IV. INDEX STRUCTURE

The base structure of the OST-tree is that of the TPR-tree for indexing the spatio-temporal data. However, in order to specify the authorization and the degree of accuracy of user’s position and time, the node structure will be modified to attach more information. Specifically, in addition to the tpbr, each node contains a pointer \( p \) pointing to the list of entries. Each entry has the form of a 4-tuple \(<id_{sp}, id_{user}, \Delta s, \Delta t>\), indicating that a service provider with the identity \( id_{sp} \) can access sensitive information of a user with the identity \( id_{user} \) at the degree of accuracy of user’s position and time specified by the value \( \Delta s \) and \( \Delta t \), respectively. Fig. 1 illustrates the structure of the OST-tree. For the illustration purpose, the values of authorizations \( \alpha_i \) (i=1..5) in this figure are \( \alpha_1 = <#S101, #U232, 1000m^2, 3m> \), \( \alpha_2 = <#S101, #U134, 600m^2, 3m> \), \( \alpha_3 = <#S102, #U232, 500m^2, 3m> \), \( \alpha_4 = <#S101, #U135, 550m^2, 4m> \), and \( \alpha_5 = <#S103, #U232, 0m^2, 0m> \).

Our goal is to develop an index structure that can incorporate the accuracy degree of user’s position. Therefore, this accuracy degree parameter must be in the hierarchical form. The OST-tree achieves this hierarchy well. Since the tpbr in a TPR-tree is already organized in hierarchical structure, the OST-tree inherits this property to hierarchically organize the bounding rectangle containing the user’s exact position that will be returned to the service providers. More specifically, when traversing from the root node to a leaf node in the OST-tree, the degree of accuracy of user’s position increases because the area of the bounding rectangle is smaller and vice versa. For example, in the traversal path N1-N5-N14 (see Fig. 1), the areas of the returned rectangles reduce from 1000m² to 500m² and 0m² corresponding to \( \alpha_1, \alpha_3, \) and \( \alpha_5 \). This means that the degree of accuracy of user’s position increases.

Based on this property, if service providers have a higher level of trust from a user, their identities will be placed on the node nearer to the leaf node and vice versa. For instance, the service provider with the identity \#S103 has the highest level of trust from a user with the identity \#U232, and so it can obtain the user’s exact position (\( \Delta s=0 \)). This service provider’s identity is, therefore, placed on the leaf node.

A. Privacy Information Overlaying and Insertion

The privacy information overlaying and insertion process happen in parallel. We traverse the OST-tree from the root node down to the leaf node to place the new object in the suitable leaf node (by applying the insertion algorithm as shown in [5,8]) and, at the same time, recursively compare the degree of accuracy of user’s position (\( \Delta s \)) with a spatial extent of each node (\( \Delta s \)) in the insertion path to find the appropriate node overlaying privacy information. We have two possible scenarios for this comparison:

- Case 1: If (N is the appropriate sub-tree) and (\( \Delta s \leq \Delta s \)), we overlay \( \alpha \) on N and continue the insertion process.
- Case 2: If (\( \Delta s > \Delta s \)), depend on the level of N, we have two scenarios: If N is a non-leaf node, we choose an appropriate sub-tree rooted at N (complying with the algorithm ChooseSubtree of R*-trees [8]) and continue the overlaying process. If N is a leaf node, we overlay and insert the new object into this node.

If a moving object has already existed in the index structure and the user wants to add new policies, we find the appropriate node in the insertion path to overlay privacy information.

Since the authorization is put as high as possible in the OST-tree, the search process can stop at some internal node if the match occurs. Thus, we do not always have to traverse to the leaves to find a user’s exact position as in algorithms separated from the database level. For example, if the service provider \#S101 wants to obtain the position of user \#U135, the search process stops at the internal node N6-N7 and returns the result. But, in the case of an algorithm separated from the database level, we have to traverse to the leaf node N15, where
the position of #U135 belongs to, to retrieve the user’s exact position and then obfuscate it.

B. Privacy Analysis

Adversary model: the adversary tries to manipulate the obfuscated region to infer the user’s exact location. For obfuscation techniques, the relevance [11] is used to measure the location privacy protection. The lower the relevance, the higher the location privacy protection is, and thus the lower the probability an adversary can infer the user’s exact location. So, in order to analyze the location privacy protection of our proposed approach with that of the approach that separates the algorithm from the database level, we will compare the relevance values of the two approaches.

For the approach that separates the algorithm from database level, the relevance is:

\[ R_t = \frac{(A_t \cap A_s)^2}{A_t A_s} \]  \hspace{1cm} (5)

where \( A_t \) is the location measurement [11] and depends completely on the positioning technology, and \( A_s \) is the obfuscated region created by the privacy-preserving algorithm.

To calculate the relevance of our proposed approach, we can simply replace \( A_t \) by \( \Delta s \) in (5). However, the concept of relevance in [11] only concerns about spatial privacy protection. By taking into account the temporal element, we extend the relevance concept to use for both spatial and temporal privacy protection as follows:

\[ R_s = \frac{(A_t \cap \Delta s)^2}{A_t \Delta s} \cdot \frac{1}{\Delta t} \]  \hspace{1cm} (6)

where \( \Delta s \) and \( \Delta t \) are the degree of accuracy of user’s position and time, respectively.

From (5) and (6), we can see that \( R_s \geq R_t \) (since \( A_t = \Delta s \) and \( \Delta t \geq 1 \)), meaning that the degree of privacy protection of our proposed approach is higher than that of approach separating the algorithm from database level. More specific, incorporating the temporal dimension into the relevance concept reduces the probability that an adversary can infer the user’s exact location because the adversary has to guess not only where, but also when the user’s exact position belongs to.

C. Performance Analysis

In this section, we compare the performance between the TPR-tree and OST-tree in terms of the number of disk accesses. For the analysis, let us suppose that \( n \) is the number of moving users; \( t \) is the size of each tpr; \( d \) is the disk block size; \( M \) is the maximum number of tree pointers in one node; \( P_t \) is the size of block pointer pointing to a subtree; \( P_s \) is the size of authorization pointer pointing to the list of authorizations; \( a \) is the average number of authorization placed in each node; \( S_a \) is the size of each authorization \( a \).

For the TPR-tree, an internal node contains only time-parameterized bounding rectangles and block pointers; these must fit into a single block:

\[ M (P_t + t) \leq d \Rightarrow M \leq \frac{d}{P_t + t} \]

So, the number of disk accesses is:

\[ \Omega(\log_{P_t + t} n) \]  \hspace{1cm} (7)

For the OST-tree, we have two cases: the list of authorizations is pointed by a pointer or embedded directly into the nodes. For the first case, an internal node of OST-tree contains the time-parameterized bounding rectangles, block pointers and an authorization pointer; these must fit into a single block:

\[ M (P_t + t) + P_s \leq d \Rightarrow M \leq \frac{d - P_s}{P_t + t} \]

So, the number of disk accesses is:

\[ \Omega(\log_{P_t + t} n) \]  \hspace{1cm} (8)

For the second case, an internal node will have the same number of time-parameterized bounding rectangles and block pointers, but the size of authorization depends on the number of authorization placed in each node. Hence, the order \( M \) can be calculated as follows:

\[ M (P_t + t) + a S_a \leq d \Rightarrow M \leq \frac{d - a S_a}{P_t + t} \]

So, the number of disk accesses is:

\[ \Omega(\log_{P_t + t} n) \]  \hspace{1cm} (9)

In OST-tree, if the authorization is embedded directly into the node, it will require less disk accesses than that of the first case where the authorization is pointed by a pointer.

Also, the above analysis shows that when traversing to leaf nodes is required in two indexes, the TPR-tree has the lower height and requires less disk accesses than that of the OST-tree since the OST-tree has to reserve the space to store the authorization in each node. However, in most cases, we do not have to traverse to the OST-tree leaves to retrieve the result. Because if the pair value \(<id_{tpr}, id_{user}>\) of the query’s authorization is matched with that of some internal node, we will stop at this node and return the result without further traversing on the OST-tree. Hence, the OST-tree requires less disk accesses than that of the TPR-tree. Only in the worst case where users are willing to reveal their exact position to service providers, we have to traverse to the leaf node to retrieve the exact result. For example, if the service provider #S101 wants to get position of the user #U134, the query needs visiting only two nodes (root node and N1) instead of three nodes, thereby reducing the number of disk accesses comparing to TPR-trees.

V. PERFORMANCE EXPERIMENTS

To conduct the experiments, we use the open source implementation of TPR-trees called SaIL [9]. Both TPR-tree
and OST-tree are implemented in C++, and all the experiments are conducted on a Core 2 Duo Personal Computer with 1 GB of memory. In the experiments, we use uniform data, where object positions are randomly generated and speeds ranging from 0.25 to 1.66 are chosen at random. The effective fill factor is usually close to 70%. The fan-out of internal and leaf nodes is 20 with a 4K page size. The maximum update interval is 20. The number of query is 35 and the horizon time is 20.

Since the node of OST-tree contains authorizations, the number of records in each OST-tree’s node is smaller than that of the TPR-tree. So, the OST-tree requires more nodes to contain the same number of moving objects (cf. Fig. 2).

By incorporating the time into the privacy model, the average relevance of our proposed approach (Rst) is smaller than that of the obfuscation algorithm (Rs) (cf. Fig. 3).

Fig. 4 and Fig. 5 compare the insert cost between the TPR-tree and OST-tree in terms of CPU time and number of I/O operations, respectively. The insert cost of OST-tree is higher than that of TPR-tree since we have to spend extra time (or number of I/O operations), besides the time for insertion process, to find appropriate node to overlay authorization.

Given the mobility of users, the update cost as shown in Fig. 6 of OST-tree is higher than that of TPR-tree, because OST-tree has to incur the additional cost of updating the authorization (moving α from current node to another node corresponding to the newly updated position of a user).

Fig. 7 compares the query cost between the two indexes in terms of the number of I/O operations. The query in this case is point location queries, which retrieves the rectangle containing the location of user. In general, the query cost of OST-tree is better than that of TPR-tree since we do not have to traverse to the leaf node to get the result in OST-tree. Only in some cases, where users want to reveal their exact locations to service providers, the OST-tree is not better than TPR-tree, because we have to traverse to the leaf nodes of OST-tree. Hence, OST-tree is better than TPR-tree in cases users just want to reveal a low degree of accuracy of their locations to service providers.

VI. CONCLUSION AND FUTURE WORK

In this work, we have introduced the OST-tree capable of obfuscating the spatio-temporal data of users. Although the OST-tree requires more storage space and update overhead, it achieves the lower querying cost and higher privacy protection comparing to the TPR-tree.

Future work will extend the probability distribution of user’s position so that the probability that a user’s position (x, y) belongs to a region is not uniformly distributed. Because, in real life, the region where a user belongs to depends on many factors related to geography, it is easy for the adversary to infer a user’s exact position in the obfuscated area if the probability distribution of user’s position is uniformly distributed.

REFERENCES


