B^{ob}-Tree: An Efficient B^{+}-Tree Based Index Structure for Geographic-Aware Obfuscation

Quoc Cuong To[1], Tran Khanh Dang[1], and Josef Küng[2]

[1] Faculty of Computer Science & Engineering, HCM University of Technology, Vietnam
{qcuong, khanh}@cse.hcmut.edu.vn
[2] FAW Institute, Johannes Kepler University Linz, Austria/Europe
josef.kueng@faw.jku.at

Abstract. The privacy protection of personal location information increasingly gains special attention in the field of location-based services, and obfuscation is the most popular technique aiming at protecting this sensitive information. However, all of the conventional obfuscation techniques are geometry-based and separated from the database level. Thus, the query processing has two time-consuming phases due to the number of disk accesses required to retrieve the user's exact location, and the location obfuscation. Also, since these techniques are geometry-based, they cannot assure location privacy when the adversary has knowledge about the geography of the obfuscated region. We address these problems by proposing B^{ob}-tree, an index structure that is based on B^{dual}-tree and contains geographic-aware information on its nodes. Experiments show that B^{ob}-tree provides a significant improvement over the algorithm separated from the database level for query processing time and location privacy protection.

Keywords: LBS, obfuscation, privacy-preserving, spatio-temporal indexing.

1 Introduction

With the rapid development of mobile technologies, there are more than 4.5 billion mobile users by the year 2009 and the number is expected to increase more. Among 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 location. 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. We consider location privacy as an enabling technology for the proliferation of LBS, and so must balance the privacy and service quality.

To solve this privacy-preserving problem, many techniques have been suggested and the most popular one is obfuscation [1,2,3,4]. The general idea of this technique is to degrade the quality of user's location information but still allow them to use services with acceptable quality. However, this technique has two major limitations. First, all of the proposed obfuscation algorithms are geometry-based. In other words, they do not consider the geographic feature constituting the obfuscated region (e.g., a lake in an obfuscated area). Based on the knowledge of the region geography, an

N.T. Nguyen, C.-G. Kim, and A. Janiak (Eds.): ACIIDS 2011, LNAI 6591, pp. 109–118, 2011.

adversary can increase the inference probability of a user's exact location. Second, these algorithms are separated from the database level, making the algorithms go through two time-consuming phases due to the number of disk access required to (1) retrieve user's exact location on the database level, and (2) obfuscate this information on the algorithm level. Also, it is prone to privacy violation and more deployment complexity because both phases, together with the communication channel between them, must be trusted.

Motivated by these reasons, in this work, we create a new geographic-aware obfuscation technique and propose B^{ob}-tree, a new spatio-temporal index structure based on B^{dual}-tree [8]. By taking into account the geographic feature inside the obfuscated region, our new technique ensures a higher privacy protection degree than that of the geometry-based obfuscation techniques in [1,2,3,4]. Furthermore, because B^{ob}-tree embeds geographic-aware region information on its nodes, the process of calculating the obfuscated region can be done in only one phase: traversing the index structure to retrieve the appropriate *obfuscated* region that contains a user's exact location. This one-phase process can reduce the processing time considerably comparing to the two-phase process mentioned above.

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 presents the new geographic-aware obfuscation technique. Section 4 introduces B^{ob}-tree. Section 5 gives privacy and performance analyses. Section 6 presents experimental results, and section 7 gives concluding remarks.

2 Related Work

2.1 Location Obfuscation

Among the most popular techniques to protect user's location privacy, obfuscation based techniques have gained much interest due to its intuition and implementation simplicity. Location obfuscation aims at hiding user's exact location by decreasing the quality of user's location information. In [1], Ardagna et al. propose obfuscation techniques by enlarging the area containing user's real location. However, these techniques just deal with geometry of the obfuscated region, not concerning about what is included inside (i.e., the geographic feature). Of late, the semantic-aware obfuscation technique introduced in [13,14] considers sensitive feature types inside an obfuscated region. But, this technique does not concern about how big the area of the obfuscated region is. It focuses only on the probability that a user is located in a sensitive place. In various LBS, however, an indispensable requirement is that the area of any obfuscated region must be big enough to protect user's location privacy.

With obfuscation techniques, 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 users to decide what location accuracy degree to be revealed to which service providers. Inspired by this, in [9], Dang et al. introduce an architecture to classify the service providers depending on the user's trust. This architecture inherits the property of mandatory access control to label each service provider so that users only reveal their locations on an appropriate level based on the labels assigned to the service providers. Similar to this idea, our

proposed approach classifies service providers in the way that the more reliable the service providers, the smaller area of the obfuscated region they can obtain.

2.2 Spatio-temporal Structures for Indexing Moving Objects

A number of recent researches focus on indexing the present and future positions of moving objects [5], and the two most dominant popular methods are *parametric spatial access* and *space-filling-curve transformation*. With the former, the main idea is that the bounding rectangle is a temporal function, and thus can enclose moving objects. The most popular access method in this category, TPR-tree [6], inherits the idea of parametric bounding rectangles in R-tree [15] to create time-parameterized bounding rectangles (TPBR). However, the TPBR bear two crucial limitations that dramatically affect the performance of TPR-tree: overlapping and high storage cost. The latter overcomes these two limitations by using the space filling curves (e.g., Peano/z-order, Hilbert) to transform object locations from multi-dimension to one-dimension space. Then, these one-dimensional values are indexed by a B^+-tree, which is the typical one-dimensional index. Two most popular access methods in this category is B^x-tree [7] and B^{dual}-tree [8]. The B^x-tree outperforms the TPR-tree by factors of as much as 10 but it fails to consider object velocity, and thus the query processing with B^x-tree retrieves a large number of false hits, which seriously affects its performance. B^{dual}-tree overcomes this limitation by capturing also the velocity information. By using the partitioning grid that divides the data space into cells, B^{dual}-tree can effectively answer progressive spatio-temporal queries which are poorly supported by B^x-tree.

Despite the existence of several indexing techniques for present and future positions, to the best of our knowledge, no moving-object index has yet been reported in the literature that achieves the goal of obfuscating the geographic-aware region.

2.3 Access Methods for Privacy-Preserving

All of existing privacy-preserving algorithms are separated from the database level [1,2,3,4,13]. This separation, as mentioned above, makes the two-phase query processing time-consuming. Motivated by this, Atluri et al. [10] create S^{STP}-tree, a unified index structure that embeds users' profile vectors directly into its nodes, to support profile conditions. The limitation of this access method is that it only allows or denies the access request of subjects, but does not concern about obfuscating the user's location. In other words, the access request evaluation has only two levels of result: reject or accept. Our proposed index structure, however, has multi-level form of result as evaluating an access request, based on the user's trust in service providers.

Very recently, the OST-tree [11] embeds the user's privacy policy into its nodes and obfuscates spatio-temporal data. But, since OST-tree is based on TPR-tree and concerns only with geometry-based obfuscation, it has high storage cost and quite low privacy protection.

It is evident from the above discussions that currently there does not exist any spatio-temporal index structure that can effectively handle geographic-aware obfuscation. Again, all of them are based on TPR-tree which is much less efficient than B^{dual}-tree in terms of storage cost and query processing time [8]. Towards this

goal, in this paper, we propose the B^{ob}-tree, a structure originally based on B^{dual}-tree, but with essential modifications to support geographic-aware obfuscation.

3 Geographic-Aware Obfuscation

As discussed above, although there exist a variety of research activities in spatial obfuscation, none of the proposed techniques concern with the geographic features. This can leave a backdoor to privacy open as the adversary has the geography knowledge of the obfuscated region. To address this problem, in this section, we present a new geographic-aware obfuscation technique that takes into account both the area of and the geographic feature inside the obfuscated region. This newly proposed technique not only ensures the same quality of service as others as in [1,2,3, 4] (because the obfuscated regions produced by these techniques have the same area), but also has better user's location privacy protection (as proved in section 5.2).

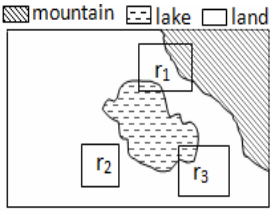

Fig. 1. Example of unapproachable region

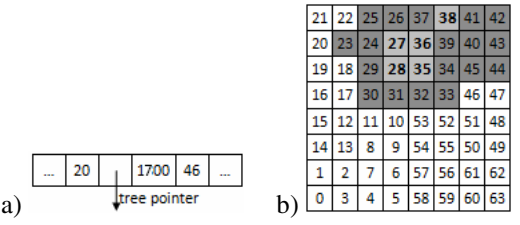

Fig. 2. A part of internal node of B^{ob}-tree and its projection on coordinate space

In our proposed technique, the region is divided into two geographic features: *approachable* and *unapproachable* parts. The unapproachable parts represent places where users, because of some reasons, cannot enter. In contrast, users can enter approachable parts. For example, the lake and mountain are the unapproachable parts of the region in Fig. 1 because no boats are allowed on the lake and user cannot climb the mountain. Our proposed obfuscation technique returns to service providers the obfuscated regions that not only have the same area as that of geometry-based techniques, but also contain only approachable features inside it.

Adversary model: Using the external knowledge of the geographic feature inside the obfuscated region, the adversary tries to eliminate the unapproachable parts of the obfuscated region, leaving only the approachable parts. In this way, the area of the original obfuscated region created by the previously proposed algorithms, e.g., as in [1,2,3,4], will be reduced since the returned region, in this case, includes both the approachable and unapproachable parts. As a result, the probability that an adversary, with his external knowledge, can infer the user's exact location within the obfuscated region is higher. The adversary, however, cannot reduce the area of the region created by our newly proposed geographic-aware obfuscation technique because this returned region includes only the approachable parts. Thus, for the two techniques, although the areas of the two regions are the same, the region created by our proposed

technique achieves better location privacy protection. For example, in Fig. 1, since the region r_1 contains two unapproachable parts (a mountain and a lake), the adversary can reduce r_1 to a smaller region by eliminating the intersection of r_1 with the lake and mountain. The region r_2, however, does not intersect with the lake or mountain, and so it is impossible for the adversary to reduce this region.

4 Index Structure

The base structure of the B^{ob}-tree is originated from that of the B^+-tree which indexes the one-dimensional values. Similar to B^{dual}-tree [8], a d-dimensional moving point o in our index structure with a reference timestamp $o.t_{ref}$, d coordinates o[1],…,o[d], and d velocities o.v[1],…,o.v[d] has its dual in the 2d-dimensional vector as follows:

o^{dual}=(o[1](T_{ref}),…,o[d](T_{ref}),o.v[1],...,o.v[d]), where o[i](T_{ref}) is the i-th coordinate of o at time T_{ref} and is given by: o[i](T_{ref})=o[i] + o.v[i]*(T_{ref}-$o.t_{ref}$)

This 2d-dimensional point in a dual space is mapped to an one-dimensional value using Hilbert curve, and then this value is indexed by B^+-tree. However, in order to specify the geographic-aware region, the node structure is modified to attach this information. Specifically, beside the one-dimensional Hilbert value transformed from the corresponding multi-dimensional point, each internal node contains the area of the approachable regions corresponding to the Hilbert range of the node.

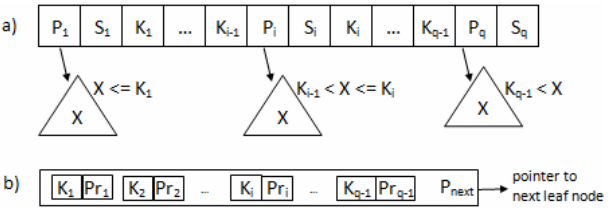

Fig. 3. B^{ob}-tree

Fig. 3 illustrates the structure of the B^{ob}-tree. Each internal node is of the form <P_1,S_1,K_1,P_2,S_2,K_2,…P_{q-1},S_{q-1},K_{q-1},P_q,S_q> where P_i is the tree pointer, S_i is the area of the approachable regions associated with a Hilbert interval [K_{i-1},K_i], where K_i is the search key value. Each leaf node is of the form <<K_1,Pr_1>, <K_2,Pr_2>, …<K_{q-1},Pr_{q-1}>, P_{next}> where Pr_i is a data pointer, and P_{next} points to the next leaf node of the B^{ob}-tree.

In B^{dual}-tree, each internal node e is implicitly accompanied by an interval [$e.h^l$, $e.h^u$], where $e.h^l$ and $e.h^u$ are Hilbert values of the starting and ending cells of the region represented by node e, and thus e is associated with $e.h^u$-$e.h^l$ cells. The area of a region associated with each internal node is then calculated by multiplying the total number of cells of each internal node with the area of the projection of each cell into the coordinate space. However, the region associated with $e.h^u$-$e.h^l$ cells includes both approachable and unapproachable regions. Thus, in order to increase the privacy degree of the region, we must filter out all unapproachable portions in this region.

Assume that the projection result of a region associated with an internal node e (associated with $e.h^u$-$e.h^l$ cells in the 2d-dimensional space) into the coordinate

space is the region consisting of $e_1.h^u$-$e_1.h^l$ cells (in the 1d-dimensional space), and there are x unapproachable cells within this 1d-dimensional region. Obviously, the number of approachable cells associated with e is $e_1.h^u$-$e_1.h^l$-x. Thus, the area of approachable regions associated with e is ($e_1.h^u$-$e_1.h^l$-x)S_c, where S_c is the area of each cell in the 1d-dimensional space. For example, Fig. 2a shows an internal node and its associated Hilbert value transformed from the 4-dimensional space. Fig. 2b is the projection of this node into the 2-dimensional coordinate space. The five gray cells 27, 28, 35, 36, and 38 are unapproachable. Assume that area of each cell is $100m^2$, the area of approachable regions associated with this internal node is (45-23-5)x100=$1700m^2$, where two values 23 and 45 are the Hilbert values of projection into the 2-dimensional space of the two cells 20, 46 in the 4-dimensional space.

The authorization α used in our approach is a 3-tuple $<id_{sp}, id_{user}, \Delta s>$ where id_{sp} is the identity of the service provider, id_{user} is the user's identity, and Δs is the area of the approachable region. The meaning of an authorization is that a user id_{user} only allows the service provider id_{sp} to access his/her sensitive personal location information with an accuracy degree of Δs. For example, the user #U232 is willing to reveal his position in an approachable region, with the area of $600m^2$, to the advertising service #S101. This authorization can be expressed as α_1 = <#S101, #U232, $600m^2$>. If the user's exact position is at coordinate $<x_0, y_0>$, the result returned to the service provider is an approachable region of $600m^2$, containing the coordinate $<x_0, y_0>$.

The area of an obfuscated region associated with each node in a B^{ob}-tree is hierarchical because the interval [$e_1.h^l$,$e_1.h^u$] is smaller when traversing from the root to the leaf nodes. Therefore, when traversing from the root down in a B^{ob}-tree, the accuracy of user's position increases because the area of the obfuscated region is smaller and vice versa. Based on this basic property, if a service provider that has a low trust level from a user wants to retrieve the user's location, the search process can stop at some internal nodes that may be close to the root.

Search, Insertion, Deletion and Update with B^{ob}-tree. The following algorithm outlines the procedure to search for a record in B^{ob}-tree.

```
Algorithm Search
Input: a dual vector of a moving point o^dual, area of an
obfuscated region S.
Output: the region that its area equals S and contains
a moving point with a dual vector o^dual.
Transform o^dual into the Hilbert value h
while (n is not a leaf node) do
    Search node n for an entry i such that K_{i-1} < h ≤ K_i
    if S = S_i then
        return the region corresponding to the Hilbert
        interval [K_{i-1};K_i]
    else if S > S_i then
        return ExtendCell(K_{i-1},K_i,S)
    else
        n ← n.P_i //the i-th tree pointer in node n
Search leaf node n for an entry (K_i,Pr_i) with h=K_i
if found then retrieve the user's exact location
else the search value h is not in the database
```

The algorithm *ExtendCell(K_i,K_j,S)* extends the region corresponding to the Hilbert interval [K_i,K_j] by adding more *approachable* cells until the area of the extended region equals S. This ensures that the obfuscated region produced by our technique has the same area as that of the geometry-based techniques and achieves better location privacy protection because it contains only approachable features.

In this search algorithm, if the area S is big (e.g., the service provider gets a low trust level from the user, and thus can only obtain a big region containing the user's exact location), the search process can stop at some internal node near the root. In this case, the disk access number is reduced significantly. So, we do not have to traverse to the leaf node to find the user's exact position as in the previously proposed algorithms, which are separated from the database level. With our approach, only when users are willing to reveal their exact location to service providers, the search process must traverse to leaf nodes.

The insert, delete and update operations of B^{ob}-tree are similar to those of B^+-tree. However, since these operations change the key value in each node, the area of the obfuscated region associated with each node needs to be re-calculated.

5 Privacy and Performance Analyses

5.1 Privacy Analysis

For obfuscation techniques, the *relevance* [1] 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 real location.

$$R_s = \frac{(A_i \cap A_f)^2}{A_i . A_f} \tag{1}$$

where A_i is location measurement [1] and depends completely on the used positioning technology, and A_f is the obfuscated region area. With the algorithms separated from the database level, because A_f includes both the accessible and inaccessible regions, an adversary can eliminate the inaccessible part of A_f (cf. the adversary model in section 3). We call A_{fa} the accessible region of A_f after eliminating the inaccessible (e.g $A_{fa} \leq A_f$). The relevance of A_{fa} is calculated as follows:

$$R_{sa} = \frac{(A_i \cap A_{fa})^2}{A_i . A_{fa}} \tag{2}$$

In B^{ob}-tree, since A_f includes only the accessible region, an adversary cannot reduce A_f to a smaller region. Thus, the relevance of our proposed approach is still R_s. Since $A_{fa} \leq A_f$, from (1) and (2) we have $R_{sa} \geq R_s$. This means that the location privacy protection of our proposed approach is better than that of the introduced algorithms. More specifically, by considering the geographic feature inside the obfuscated region, we reduce the probability that an adversary can infer the user's exact location.

Similar to TPR-tree, the adversary can eliminate the inaccessible parts of A_f in relevance of OST-tree. However, the temporal obfuscation [11] in this relevance compensates the inaccessible part of A_f as follows:

$$R_{st} = \frac{(A_i \cap A_f)^2}{A_i . A_f} . \frac{1}{\Delta t} \tag{3}$$

Therefore, if the area of the inaccessible parts of A_f or the temporal obfuscation is small, the relevance of B^{ob}-tree is still smaller than that of OST-tree (e.g., $R_s \leq R_{st}$.).

5.2 Performance Analysis

In this section, we compare the performance between the TPR-tree, OST-tree and B^{ob}-tree in terms of the tree height and number of disk accesses in the query processing. Let m, m', q, q' r, r' denote the average number of entries (i.e., the fan-out) at the root, internal nodes, and leaf nodes; R, R' be the total number of records being indexed; and d, d' be the depth of the B^{ob}-tree and TPR-tree, respectively. Then we have:

$$R = (m+1)(q+1)^{d-1} r \Rightarrow d = log_{q+1}\left(\frac{R}{(m+1)r}\right) + 1 \tag{4}$$

Similarly, we have:

$$d' = log_{q+1}\left(\frac{R}{(m'+1)r'}\right) + 1 \tag{5}$$

In B^{ob}-tree, since each node contains only the integers representing the search key values and areas of the approachable regions, the storage cost for each entry is low, and thus the node fan-out is high. On the contrary, in TPR-tree, each node contains the TPBR that require high storage cost; hence the fan-out is low. In other words, averagely each internal node of TPR-tree contains fewer entries than B^{ob}-tree ($m' < m$, $q' < q$, $r' < r$). So, with the same number of records ($R = R'$), from (4) and (5) we can see that the height of B^{ob}-tree is lower than that of TPR-tree ($d' > d$), resulting in the fewer number of disk accesses in the query processing of B^{ob}-tree than that of TPR-tree. Even more, since TPBR of TPR-tree may overlap, each query processing in TPR-tree may traverse multiple paths which require many disk accesses. It is not the case for B^{ob}-tree, which is based on B^+-tree and does not incur the curse of dimensionality problem [16]. Furthermore, since $d' > d$ and the TPBR requires higher storage cost than single values in B^{ob}-tree, the storage cost of TPR-tree is higher than that of B^{ob}-tree. Also, as shown in [11], OST-tree has the higher height and requires more disk accesses in the query processing than TPR-tree since OST-tree nodes have to reserve the space to contain authorizations. Overall, B^{ob}-tree is the best one among the three.

6 Performance Experiments

To conduct the experiment, we use the open source library called SaIL (A Spatial Index Library for Efficient Application Integration) [12]. The TPR-tree, OST-tree and B^{ob}-tree are all implemented in C++, and all experiments are conducted on a Core 2 Duo PC, running Windows 7 Professional with 1GB of RAM, 160GB of HDD, and the disk page size of 4KB. For all experiments, we use uniform datasets, where object positions are randomly generated and the moving speed ranging from 0.25 to 1.66 is chosen at random. The fill factor is usually close to 70%. The index node and leaf

node capacity are 20 with 4KB page size. The maximum update interval, number of query, and Horizon time are set to 20, 35, and 20, individually.

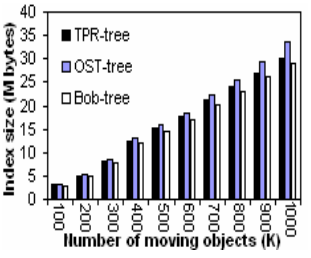

Fig. 4. Storage cost

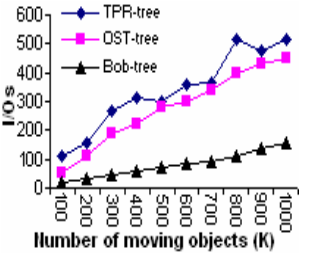

Fig. 5. Query cost (I/Os)

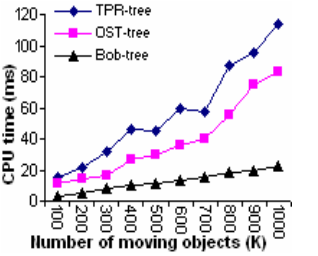

Fig. 6. Query cost (CPU time)

Beside the TPBR, OST-tree's nodes also contain authorization information, thus the number of records in each of OST-tree's node is smaller than that of the TPR-tree. So, the OST-tree requires more storage space than TPR-tree. B^{ob}-tree requires less storage space than TPR-tree since its fan-out is higher (Fig. 4). As the number of moving objects increases, the TPBR in TPR-tree and OST-tree have higher probabilities of overlapping, and the height of B^{ob}-tree is lower than that of TPR-tree and OST-tree. The query cost of B^{ob}-tree is lowest (Fig. 5 and Fig. 6).

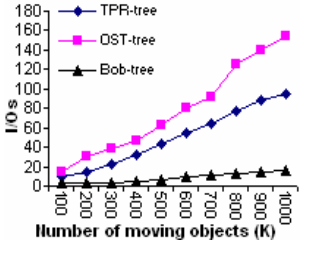

Fig. 7. Update cost (I/Os)

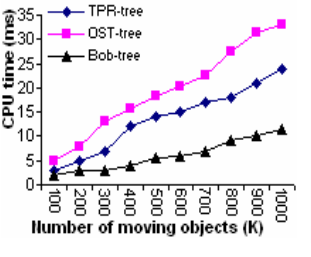

Fig. 8. Update cost(CPU time)

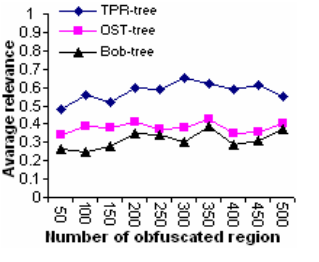

Fig. 9. Relevance

Fig. 7 and Fig. 8 show that the update cost in B^{ob}-tree achieves considerable improvement over TPR-tree and OST-tree because in B^{ob}-tree, given the key, an update needs to traverse only one path. On the contrary, in TPR-tree and OST-tree, an update may traverse multiple paths due to the overlaps among TPBR. As the dataset grows, more overlap happens and thus results in a higher update cost. In Fig. 9, by considering the geographic features inside the obfuscated region, the average relevance of B^{ob}-tree (R_s) is much smaller than that of the geometry-based obfuscation algorithms (R_{sa}). The relevance of B^{ob}-tree is slightly smaller than that of OST-tree (R_{st}) since temporal obfuscation compensates the inaccessible parts.

7 Conclusion and Future Work

In this work, we have created the B^{ob}-tree, based on the B^{+}-tree and B^{dual}-tree, that is capable of obfuscating the geographic-aware regions. Our novel index structure is much more efficient than both OST-tree and TPR-tree in terms of storage space, query processing time, update and privacy protection.

In the future, we will consider the use of the B^{ob}-tree to support other privacy-preserving techniques in LBS (e.g., k-anonymity), and will address the quality of LBS problem due to the different shapes of the returned regions wrt. the B^{ob}-tree and other access methods. Another research problem of interest is to extend authorization from $<id_{sp}, id_{user}, \Delta s>$ to $<id_{sp}, id_{user}, \Delta s, \Delta t, \Delta p>$, where Δt and Δp represent the accuracy degree of time and profile, in order to support temporal and profile obfuscation.

References

1. Ardagna, C.A., Cremonini, M., Vimercati, S.D.C., Samarati, P.: An Obfuscation-Based Approach for Protecting Location Privacy. TDSC 8(1), 13–27 (2009)
2. Anh, T.T., Chi, T.Q., Dang, T.K.: An Adaptive Grid-Based Approach to Location Privacy Preservation. In: ACIIDS, Hue, Vietnam, pp. 133–144 (2010)
3. Jafarian, J.H., Amini, M., Jalili, R.: Protecting Location Privacy through a Graph-based Location Representation and a Robust Obfuscation Technique. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 116–133. Springer, Heidelberg (2009)
4. Mohamed, F.M.: Privacy in Location-based Services: State-of-the-art and Research Directions. Tutorial, MDM, Germany (2007)
5. Dinh, L.V.N., Aref, W.G., Mokbel, M.F.: Spatio-temporal Access Methods-Part 2. IEEE Data Engineering Bulletin (2010)
6. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. In: ACM SIGMOD, USA, pp. 331–342 (2000)
7. Jensen, C.S., Lin, D., Ooi, B.C.: Query and Update Efficient B^{+}-tree based Indexing of Moving Objects. In: VLDB, Canada, pp. 768–779 (2004)
8. Yiu, M.L., Tao, Y., Mamoulis, N.: The B^{dual}-Tree: Indexing Moving Objects by Space-Filling Curves in the Dual Space. VLDB Journal 17(3), 379–400 (2008)
9. Dang, T.K., To, Q.C.: An Extensible and Pragmatic Hybrid Indexing Scheme for MAC-based LBS Privacy-Preserving in Commercial DBMSs. In: ACOMP, Ho Chi Minh City, Vietnam, pp. 58–67 (2010)
10. Atluri, V., Shin, H.: Efficient Security Policy Enforcement in a Location Based Service Environment. In: DBSEC, USA, pp. 61–76 (2007)
11. To, Q.C., Dang, T.K., Küng, J.: OST-tree: An Access Method for Obfuscating Spatio-temporal Data in Location-based Services. In: NTMS, France (to appear, 2011)
12. Hadjieleftheriou, M., Hoel, E., Tsotras, V.J.: SaIL: A Spatial Index Library for Efficient Application Integration. Geoinformatica 9(4), 367–389 (2005)
13. Damiani, M., Bertino, E., Silvestri, C.: Protecting Location Privacy through Semantics-aware Obfuscation Techniques. In: IFIPTM, Norway, pp. 231–245 (2008)
14. Damiani, M., Bertino, E., Silvestri, C.: PROBE: an Obfuscation System for the Protection of Sensitive Location Information in LBS. TR2001-145, CERIAS (2008)
15. Guttman, A.: R-trees: A Dynamic Index Structure for Spatial Searching. In: ACM SIGMOD, USA, pp. 47–57 (1984)
16. Dang, T.K., Küng, J., Wagner, R.: The SH-tree: A Super Hybrid Index Structure for Multidimensional Data. In: Mayr, H.C., Lazanský, J., Quirchmayr, G., Vogel, P. (eds.) DEXA 2001. LNCS, vol. 2113, pp. 340–349. Springer, Heidelberg (2001)