Privacy Preserving through A Memorizing Algorithm in Location-Based Services

Quynh Chi Truong, Anh Tuan Truong, Tran Khanh Dang

Faculty of Computer Science & Engineering, HCMUT
National University of Ho Chi Minh City, Vietnam
{tqchi, anhtt, khanh}@cse.hcmut.edu.vn

ABSTRACT
Along with the rapid development of mobile devices having embedded positioning technology, location-based services have become more and more popular. In order to use the services, users often disclose some parts of their private information, especially their location-related information. Thus, it raises a great concern in the users' privacy preserving in location-based services. This paper proposes a solution with a memorizing algorithm working on a trusted middleware. With the proposed solution, the space is organized in a flexible grid and the middleware cloaks the user's location information in an anonymization area before sending it to the service providers. A concerned problem is that overlapped areas among anonymization areas can be used to explore the true position of a user because the overlapped areas have a higher probability of having a user. Our newly introduced memorizing algorithm calculates on the spatial grid to decrease the overlapped areas as much as possible. The solution aims at protecting the user's privacy not only at the time using the service but also against data mining techniques wrt. their history location data. Experimental results with synthesis moving objects over real world maps will establish our theoretical analyses as well as the practical value of the proposed solution.

General Terms

Keywords
Location-based Services, Location Privacy, Privacy Preserving, Memorizing Algorithm, Privacy Preserving in Data Mining.

1. INTRODUCTION
Advances in location technologies and wireless communication technologies enable the widespread development of location-based services (LBS) that make use of the location information of users [16, 22]. As location information is a part of users’ private information, it requires a number of solutions to protect the location privacy of users while not affecting much on the quality of the location-based services. Location privacy can be defined as the right of individuals, groups, and institutions to determine themselves how, when, to whom, and for which purposes their location information is used [2, 16, 19]. When the user location information is not well protected, the user can face various kinds of location attacks. Some attacks just make the user annoying, for instance unconsenting advertisements, while the others can endanger the user such as stalking or physical harassment [2, 3, 8]. The problem of privacy preserving in LBS attracts numerous attentions from both research communities and industry sectors [8]. The user’s location privacy should be safeguarded in two stages. In the first stage, the location privacy should be protected at the time of using services. One popular method is to obfuscate the location with the service providers in order to hide the user’s true location information [19]. The solution focuses on preventing the user’s locations from an illegal observation at the time of service calls. However, when a user uses the service several times in a specific area, it will cause an overlapping problem which can be exploited to identify the highest possible area where the user is [14]. Then, it leads to the second stage which ensures the user’s privacy when the user’s location information is stored in the database for data mining purposes [14].

Although there are many researches on this field, they only concentrate on privacy preserving in either the first stage or the second stage. This paper proposes a novel approach for privacy preserving in both stages. Our solution bases on a LBS framework consisting of a trusted middleware (see figure 1(b)). We also introduce an algorithm that applied in the middleware. The algorithm has two main functions: to cloak the user’s location and to reduce the area of the overlapped region.

The rest of this paper is organized as follows. In section 2, we briefly summarize the related work. Section 3 presents our discussion on the privacy problem of overlapped areas. Next, section 4 presents our grid-based approach for the problem. Section 5 introduces our memorizing algorithm that works on the grid for preserving privacy in LBS. Experimental results are shown in section 6. Finally, section 7 presents concluding remarks as well as future work of our solution.
2. RELATED WORKS

In the scope of LBS, there are three system architectures for preserving location privacy: the non-cooperative architecture, the centralized trusted party architecture, and the peer-to-peer cooperative architecture [19]. In the first architecture, users are self-responsible for protecting their location privacy. The users can provide false identities or location to the service providers. They can also create many dummies to hide the true one. This is an easy way to protect location privacy but the critical foible is that it totally depends on users’ knowledge.

In the second architecture, there is a trusted party which stands in the middle between the users and the service providers. The trusted party can be a third party server or a middleware [4]. The main duty of the trusted party is to provide a location transparency mechanism. A location transparency mechanism is defined as hiding all aspects of location information from the service providers, including location values, and positioning methods [16]. First, it receives the location information from the user, blurring and sending the information to the service providers. Then, it filters and forwards back the results to the user. There are several algorithms that can be applied in the trusted party, namely k-anonymity [11, 15, 19], k-area cloaking [2, 18, 19], mix zone [2, 5, 6, 18, 19], and so on. This architecture fulfills the weak point of the first architecture because it does not rely on users’ awareness. Moreover, the architecture is flexible as it separates functional module (the service providers) and the privacy module (the trusted party). However, the disadvantages in this architecture are bottle-neck problem and how trust the third party is [19].

In the peer-to-peer cooperative architecture, users gather in a group and collaborate with each other so that the service providers could not distinguish a particular user [19]. The problem is that it is not always that a user has a group.

In general, the second architecture, the centralized trusted party architecture, is the most obvious one to deploy. Therefore, in our solution, we describe an algorithm that works on the trusted party, in particular the middleware.

3. ANONYMIZATION AREA AND PRIVACY PROBLEM

In the second architecture, when a user wants to use services, he must send his true location information to the trusted party. After anonymizing the user’s location, the trusted party sends the anonymized area to the service providers. In case that the service provider is not trusted, the database of users’ location information (after anonymized) is not secret. Attackers can get control the database and freely exploit it.

In practical, the user usually uses the services in a specific area. For example, the user lives in a certain district, and uses the location-based services to find the cheapest shopping center around him many times and at different time points. Each time he calls the service, the middleware issues an anonymization area corresponding with his true location and sends to the service providers. The problem is that the middleware does not memorize all previous anonymization areas, so it yields the anonymization areas randomly. This causes an overlapped area problem when the user uses the service many times.

4. GRID-BASED SOLUTION FOR THE TRUSTED PARTY ARCHITECTURE

In this section, we will introduce an approach basing on a spatial grid. With this approach, the location of the user will be anonymized on a grid. An anonymization area includes cells and the location of the user is in one of these cells. In case that the service provider is not trusted, the database of users’ location information (after anonymized) is not secret. Attackers can get control the database and freely exploit it.

In practical, the user usually uses the services in a specific area. For example, the user lives in a certain district, and uses the location-based services to find the cheapest shopping center around him many times and at different time points. Each time he calls the service, the middleware issues an anonymization area corresponding with his true location and sends to the service providers. The problem is that the middleware does not memorize all previous anonymization areas, so it yields the anonymization areas randomly. This causes an overlapped area problem when the user uses the service many times.

4.1 Definitions

In the grid-based solution, we use a grid to divide the space into cells. The size of a cell is pre-defined. The shape of the cell can
be either a square or a rectangle but all of the cells must cover the whole space.

![Figure 3. The grid and an anonymization area](image)

An anonymization area is an area that consists of a number of cells. It is used to obfuscate the user’s true location information. In the figure 3, the gray area represents an anonymization area of the true location S of the user.

We also define a privacy level as the level that the user wants to blur his true location information. The higher the level is, the larger the anonymization area is, and the more privacy the user has. The privacy level p corresponds with a square-shaped anonymization area that contains p*p cells. For example, the anonymization area in the figure 3 has 16 cells (4*4 cells) with the privacy level 4.

In this approach, the grid needs a starting point. Depending on the real map, we can choose a proper starting point. Then, we can identify a certain cell by computing the distance between this cell and the starting point.

**4.2 Architecture**

First, we will review the second architecture which has a trusted middleware. In the following parts, we call the trusted middleware as middleware for short. When the user wants to use the services, he sends his true location information to the middleware with the required privacy level. When the middleware receives this information, it embeds the location information into an anonymization area according to the privacy level. After that, it sends the anonymization area to the service providers. After receiving the results from the service providers, it filters the reasonable results and sends the results to the user.

The grid-based solution also bases on the second architecture. In this architecture, the grid will be put in the middleware. The middleware will choose cells from this grid to form a rectangle according to required privacy level.

In figure 2, the user sends his true location information S to the middleware with the privacy level 4. Then the middleware finds the grid covering the space of the user’s location. It also chooses 16 cells to form a 4*4-rectangle and sends this rectangle to the service providers. The anonymization area is gray in figure 2.

With this grid, it is simple to satisfy a required privacy level of the user. When the user requires a higher level, the anonymization rectangle will be extended and when the user requires a smaller level, the rectangle will be reduced.

**4.3 Overlapping Problems and the Grid Based Solution**

In the random approach, when the user sends the true location information to the middleware, the middleware will embed this location information into a random area. If the user uses the service at different times, the middleware can send different areas to the service server. Attackers can use these areas to limit the space containing the user’s location. If the user uses services more times, the attackers have more changes to find the true location of the user.

With grid-based solution, the user can use the services many times but the smallest space which attackers can limit is a cell containing the location of the user. For example, a user A uses service at time t1 and sends his location S to the middleware. Then the middleware will send the rectangle R1 to the service server. Similarly, at time t2 and t3, R2 and R3 are sent to the service server.

![Figure 4. The overlapped area](image)

When the attackers have three rectangles, they can infer the true location of the user by getting intersection of these rectangles. The smallest area that the attackers can limit is a cell because the smallest intersection of these rectangles is a cell.

Although the trusted middleware uses the same grid to anonymize the user’s location, each user has a different requirement of the privacy level. Therefore, it is difficult to decide how large the cell is. If the size of a cell is too small, it is not enough to preserve the location privacy of the user. Otherwise, if the size of a cell is too large, it will decrease the service’s quality. To solve this problem, we can design a grid in which the cells can be resized. Another solution is to combine the cells or to split a cell to form new cells. We will consider them as future work.

Moreover, we will see an example: at time t1, the user is at S and uses the service with the required privacy level 3 (3*3 rectangles). The middleware will embed his location into R1. At time t2, the user is at S’ and also uses the service with the privacy level 3; his location is embedded into R2.

![Figure 5. A problem with grid-based solution](image)

However, the attackers can take two rectangles and find that the actual level is 2 cells because the space containing the location of the user is embedded into two cells C1 and C2. Thus, it does not satisfy the required level of the user.

In the next section, we introduce an algorithm to solve this problem. This algorithm requires that the middleware memorize the anonymization rectangles.
5. MEMORIZING ALGORITHM FOR GRID BASED SOLUTION

The two problems in the section 4.3 have the same cause. It is randomization. At different times, the middleware will create different rectangles and send to the service server. The rectangles are created randomly basing on the user’s location. Therefore, the more rectangles are created, the more accurate attackers can find the user’s location. To solve these problems, we can use a database to save the rectangles. At different times, the middleware checks the database and finds the proper rectangle(s) if any. Thus, the middleware creates only one rectangle at the first time and uses it for the next times.

In short, the mechanism of this solution is as follows:

- When the user wants to use service, he sends his true location information to the middleware and a required privacy level.
- The middleware checks the database to verify whether this is first time the user uses the service in the area or not:
  - If the answer is yes, a rectangle is created randomly according to the required privacy level. Then, the middleware sends this rectangle to the service server and also saves this rectangle to its database.
  - Otherwise, the middleware gets the rectangle returned from the database and sends this rectangle to the service server.

In this algorithm, when the middleware queries the database to find down if this is first time or not, it will need the true location information of the user. For example, at location S (see figure 6), the user uses the service and, the middleware creates the rectangle R1 and stores to its database:

At location S’, the middleware checks and finds that the user used the service in the past, but the current location S’ of the user is not in the area R1. In this case, the middleware will consider that at location S’, the user uses the service for the first time and creates a new rectangle. However, when the user is at S’’ and wants to use the service again, the middleware will find that it is not the first time the user calls the service in this area because S’’ belongs to the area R2. Therefore, the middleware will return the rectangle R2 instead of creating a new rectangle.

In summary, when the user requests the service, if this is the first time he uses the service at this position, the middleware will cloak him in a random rectangle which satisfies his required privacy level. Otherwise, it will reuse the anonymization rectangle stored in the database in the previous usage having the same privacy level. Clearly, when the user requires a same level as the first time, the middleware does not need do anything; it will return the same rectangle saved in the first time. However, the user may change his privacy level comparing to the previous call of the service. We will consider following example:

![Figure 7. Problem when users require the higher privacy level](image)

At the first time, the user is at S and wants to use the service with the privacy level 3 (3*3-rectangle). Then, the middleware creates a 3*3-anonymization rectangle R1. It also memorizes R1 for the next use. After that, he wants to use the service again at a certain location in the rectangle R1. However, his required privacy level is 4. It means that he requires a bigger anonymization rectangle. The middleware finds that this time is not the first time, so it will get the rectangle saved in the database. Because this rectangle is not big enough to satisfy the required privacy level, some cells should be added to this rectangle to meet the privacy requirement. In this step, we can get some cells randomly and add to the saved rectangle to form a new rectangle. In the figure 7, 7 cells are added to R1 to form 4*4-rectangle.

In contrary, we will also examine the situation when the user wants a smaller privacy level in same area. At the first time of using service, the rectangle R1 with privacy level 4 is returned. After that, the user wants to use service again but with the smaller level, for example, level 2. As a result, R2 is created but not randomly. It must be inside R1. In other words, it means that we will “reduce” R1 to R2 (see figure 8).

![Figure 8. Problem when users require the smaller privacy level](image)

In the next time, the user uses service again and requires the same privacy level as the second time. R1 is created inside R2. However, as we mentioned above, the combination of two rectangles can reduce the privacy level of the user. Therefore, the required privacy level of the user is not satisfied. In our example, the combination of R2 and R1 can limit the area containing S to 2 cells whereas the required privacy level is 4 cells.

As we discuss before, the reason of these problems is the random in choosing cells for a rectangle. R1 and R2 are created randomly so the intersection of them can limit the area containing S. To solve these problems, the middleware should store rectangle R2 to the database and returns R2 if the user is in the area R2 and requires the service. Similarity, when the user is in R2 and requires the smaller level, we can solve it as before. We can see it as a recursive process.
However, the saving rectangle will make the database more complicated. It raises a problem related to the service performance. We have to design a data structure which adapts to our solution, i.e., storing effectively and finding quickly the proper rectangle for the service. In addition, when the user moves to a location in $R_1$ and wants to use service again, the overlapped rectangle can limit the area as we discuss before. To avoid these cases, we can divide the rectangle $R_1$ to smaller rectangles according to the user privacy level but not overlapping. For example, we can divide $R_1$ to 4 rectangles as following:

![Figure 10. A dividing solution](image)

When the user is at $S$, the middleware will send the rectangle $R_1$ to service server, but when the user moves to $S'$, the middleware will send $R_4$.

To sum up our solution, we describe the algorithm in following pseudo code:

```pseudo
def if (first time) {
  get random rectangle covering the user's position based on the privacy level;
  save this rectangle and the privacy level;
  return this rectangle;
} else {
  if (less level){
    perform dividing function and get a proper rectangle;
    save this rectangle;
    return this rectangle;
  } else if (greater level){
    get the greatest saved rectangle;
    add some cells to this rectangle to satisfy the privacy level;
    save the added rectangle and the privacy level;
    return the added rectangle;
  } else { //equal level
    return the saved rectangle;
}
```

We can see that an efficient structure data is very important. Rectangles created by the middleware are stored to the database. For a long time, rectangles are stored to database increased. A structure data for saving these rectangles is needed, we leave it as future work.

We will also consider when the user moves on trajectory, for example, when the user is in the way back home and requests the services. Attackers can combine some factors to decide the area containing the user. The figure 11 illustrates the moving object situation:

![Figure 11. Moving user problem](image)

While moving in $R_1$, if the user uses the service, the trusted middleware always sends $R_1$ to the service server. But when the user moves outside $R_1$, the trusted server will create a new rectangle and send this rectangle to the service server. The problem is that if the attackers know the user’s maximum speed, the time when the user moves out of the rectangle $R_1$, and other relative knowledge, they can limit the area containing the user. In figure 11, the attackers can easily infer the area that may contain the user just by some simple calculations.

To solve this problem, a solution was proposed. In this solution, authors use a "time delay factor". It means that with a moving user, when the user just go out of a anonymity rectangle, the solution will delay creating a new anonymity rectangle in a couple of time in order to make attackers to have difficulties to decide the location of the user [14].

In the above pseudo code, we have discarded the moving object problem. It does not mean that the algorithm does not work on this problem. We simply define a short period of time, called $t$, as a time delay factor. If the user calls the services two or more times in $t$, the user moves from a rectangle to another rectangle, the service calls will be delayed. Otherwise, the above code is run.

### 6. EVALUATION

In this section, we show the empirical evaluation we conducted in order to establish our theoretical analyses as well as the practical value of the proposed approach. However, to make it easy, we only perform the experiments in case that the user does not change his privacy level when using the service. Moreover, in practical, the user rarely changes his user privacy level each time he uses the service. Instead of this, he often defines his default privacy level when he registers for the service.

We will measure and compare the true overlapped area issuing by our solution, the memorizing algorithm, and by a random algorithm which generate cells randomly. The true overlapped area is the overlapped area which really contains the object (see figure 12). We compare our algorithm with the random algorithm because to the best of our knowledge, there is no similar research that tries to reduce the true overlapped area as we do.
Since we want to measure the true overlapped area when the user calls services many times, we put the user in a particular area, then simulate several service calls from the user. To create the area where the user stays, we create a path (from home to work) with the assumption that the user usually travels on this way. We use the path-shaped area because it is easy to illustrate the overlapping problems. To create the dataset of paths, we use the Thomas Brinkhoff’s framework for generating network-based moving objects [9]. The network used in the experiment is Oldenburg which is about 102.96 km² [23].

Firstly, we define the grid corresponding with the Oldenburg city. The area of each cell reflects the minimum security level. This means that the larger the cell is, the more security is provided, but the less quality the service is. We exam the algorithm with the grid’s size varying from 40x40 cells, 50x50 cells to 60x60 cells. In each case, we use 20 objects corresponding with its path in 20 time units. We run this dataset for 15 times with the memorizing algorithm and 15 times with the random algorithm. For each time, we calculate the average of true overlapped area (by cells) of 20 objects.

We also apply the algorithm in the grid with the privacy level 2, 3, and 4 to diversify the results. The charts show the relationship between the average of overlapped area of 20 objects and the number of service calls. In the chart, the blue line with diamond makers represents for the average values of the memorizing algorithm while the red line with rectangle makers stands for the random algorithm. Because we examine the objects in 20 time units corresponding with 20 positions, there will be maximum 20 overlapped cells for each object.

However, with a predefined grid, including the grid size and the number of cells, and a specific privacy level, and a particular dataset, we can calculate the maximum overlapped cells for each object in the random algorithm. Therefore, in each experiment, the averages of true overlapped area yielded by random algorithm are bounded by this maximum value. Moreover, this maximum value must be less than 20 as the above reason (each object is tested with 20 positions in our experiment).

With the grid size of 40x40 cells, the maximum of the average of true overlapped area yielded by random algorithm is 11.2 cells. The average number increases in the first few times of service calls; then it reaches the maximum value. Besides, the average of true overlapped area yielded by the memorizing algorithm keeps constantly only around 4 cells (see figure 13, 14, 15).

Similarly, with the grid size of 50x50 cells, the maximum of the average of true overlapped area yielded by random algorithm is 13.1 cells. The average number also increases in the first few times of service calls before reaching the maximum value 13.1 cells. Besides, the average of true overlapped area yielded by the memorizing algorithm keeps constantly only 4 cells (see figure 16, 17, 18).

In the experiment with the grid size of 60x60 cells, the results are the same as the previous ones. The maximum of the average of true overlapped area yielded by random algorithm is 14.6 cells. (see figure 19, 20, 21).
The results show that the average of true overlapped area yielded by our algorithm remains constant while the average of true overlapped area yielded by random algorithm increases in the first few times then keeping constant. Moreover, the average of true overlapped area yielded by our algorithm is about 2.98 times less than the average of true overlapped area yielded by the random algorithm. These results confirm that our approach is reasonable in reducing the number of overlapped areas.

In general, these experimental values can change slightly depending on the dataset but the overall conclusion keeps valid.

7. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a grid based solution and a memorizing algorithm that the trusted middleware can use to anonymize the location of the user. This solution solved the overlapping problem occurring when the user uses the services many times a specific location.

The paper also proposes the solution when the user wants to change his desired privacy level to preserve privacy. However, when the user moves in a trajectory, the area contains the location of the user may be limited but this area is not smaller than a grid cell as we discuss above. To the best of our knowledge, our approach is new and the introduced memorizing algorithm over the grid-like partitioned space is among the vanguard ones that try to minimize the ability that the user’s location can be discovered.

In future, we will investigate the adaptive grid that can resize its grid cells. This grid allows the trusted middleware can create a grid according to the user’s purpose to preserve privacy. Another future work should be considered, it is the design a data structure...
that can save the rectangles. This structure can help the trusted server finds the rectangle quickly, it also helps the middleware to process the user’s requests quickly.

8. REFERENCES


