A Comparison of Spatial Generalization Algorithms for LBS Privacy Preservation

Sergio Mascetti Claudio Bettini
DICo, University of Milan, Italy

Abstract

Spatial generalization has been recently proposed as a technique for the anonymization of requests in location based services. This paper presents the results of an extensive experimental study, considering known generalization algorithms as well as new ones proposed by the authors.

1 Introduction

Location-based services (LBS) have been recently attracting a lot of interest both from industry and research. When using these services, many users may be concerned with giving up one more piece of their private information by revealing their exact location, or by releasing the information of having used a particular service. More generally, the association between the real identity of the user issuing a LBS request and the request itself as it reaches the service provider can be considered a privacy threat.

Previous works ([3, 6, 5]) showed that simply dropping the issuer's personal identification data may not be sufficient to anonymize the request. For example, the location and time information contained in the request may be used, with the help of external knowledge about the location of certain users, to restrict the group of possible issuers. A notion of spatio-temporal k-anonymity was proposed as a possible solution to guarantee user's privacy. The idea is to *generalize* the location (and time) information contained in a LBS request so that, based on that information, there are at least k potential issuers.

This technique implicitly assumes that information about the spatio-temporal position of a sufficient number of potential users of the service is available to the entity performing anonymization. The typical scenario assumes the existence of a *Location-aware Trusted Server* (LTS) that can gather this information; the LTS receives the LBS requests from the users, it performs the appropriate generalization (also hiding explicitly identifying values), and it forwards the generalized request to a *Service Provider* (SP). The answer from the SP is also routed through the LTS to be redirected to the specific user with a refined result when possible [5].

The design of a spatio-temporal generalization algorithm has two goals: i) to guarantee the user's privacy by insuring that a sufficiently large number of potential users have a spatio-temporal position contained in the released region, and ii) to preserve the quality of service by minimizing the generalization. Existing approaches to generalization try to optimize the trade-off between these two goals.

As formalized in [1], the anonymization power of generalization also depends on the assumptions about the knowledge available to the attacker. Most of recent approaches [3, 6, 5] have proposed generalization techniques that achieve anonymity even in the pessimistic case in which the attacker may acquire the knowledge about the exact location of each user. However, some of the existing generalization algorithms are subject to counterattacks if additional knowledge is assumed to be available to the attacker. In particular, in [1] it is shown that the generalization algorithms proposed in [3, 6] may fail in achieving anonymity if the knowledge of the used generalization function becomes available to the attacker. In the following, we call Γ_I*-safe* the algorithms that achieve anonymity even in the case this additional knowledge is considered, and Γ_I*-unsafe* those that only assume the availability of location knowledge.

In this paper we propose a comparison of spatial generalization algorithms involving both theoretical and empirical evaluations. In addition to considering the main algorithms previously proposed in the literature, we also describe and compare two new algorithms that are designed to address the trade-off illustrated above.

2 Algorithms for spatial generalization

In this section, we outline the generalization algorithms and, for each of them, we discuss the worst-

case time complexity; experimental results are then presented in Section 3.

2.1 Γ_I-unsafe generalization algorithms

The first generalization algorithm that appeared in the literature was named *IntervalCloaking* [3]. The idea of the algorithm is to iteratively divide the total region monitored by the LTS. At each iteration the current area q_{prev} is partitioned into quadrants of equal size. If less than k users are located in the quadrant q where the issuer of the request is located, then q_{prev} is returned. Otherwise, iteration continues considering q as the next area. In order to evaluate the time complexity of the algorithm it is necessary to make some assumptions about the data structures. In our implementation of the algorithm, we used a data structure consisting of a quadTree in which each leaf has a pointer to a user, and each internal node n stores the number of users "contained" in n i.e., the number of users stored in the subtree that has n as root. The generalization algorithm traverses the quadTree from the root to the first internal node that contains at least k users. Each iteration of the algorithm is performed in constant time and the number of iterations is bounded by the height of the quadTree. In the worst case, the height of the tree is linear in the cardinality of the set I of users. However, if users are uniformly distributed (as in the case of the experimental results that we present in Section 3) the height of the tree is logarithmic in the number of users hence the algorithm has a worst-case time complexity of $O(\log(|I|))$.

Mokbel et al. [6] propose *Casper*, a framework for privacy protection that includes a generalization algorithm. In this paper we consider the "basic" data structure[1] used by Casper i.e., a balanced quadTree in which each node has a pointer to its parent, and users are stored in leaf nodes only. Moreover, the data structure consists of a table in which each user i is associated with the leaf node that contains i. The generalization algorithm starts from the leaf node that contains the issuer of the request, and iteratively traverses the tree towards the root until an area that contains at least k users is found. At each iteration, the algorithm considers the union of the area covered by the current node n and the horizontally (vertically, resp.) contiguous area covered by its sibling node. If only one of these two joined areas contains more than k users, that area is returned; if both of them contain more than k users, the one containing the minimum number is returned; otherwise, the algorithm proceeds with the next iteration. Similarly to *IntervalCloaking*, the worst case time complexity of *Casper* is linear in the height of the quadTree. However, in this case, this height is bounded by the logarithm of the number of leaf nodes if users are uniformly distributed, and it is at most linear in the same number, otherwise.

Conceptually, one of the simplest ways to generalize a request is to compute the k Nearest Neighbor query among the users and return the MBR of the result. Following this idea, Kalnis et al. [5] propose the *nnASR* generalization algorithm that picks a random user i in the set of the $k-1$ users that are the closest to the issuer, and returns the MBR of the set containing i, the issuer, and the $k-1$ users closest to i. In our implementation of the *nnASR* algorithm we used a kd-Tree to store users' locations, making possible to compute k Nearest Neighbor queries in logarithmic expected time with respect to the number of users.

2.2 Γ_I-safe generalization algorithms

To the best of our knowledge, the first Γ_I-safe generalization algorithm was proposed by Kalnis et al. [5], and it was called *hilbASR*. This algorithm and the new ones proposed in this section, have been designed for the case of the same value of k being used for all users, while they require an extension to deal with personal k values and preserve their safety. The idea of *hilbASR* is to exploit the Hilbert space filling curve to define a total order among users' locations. A data structure is then used to store users in the order defined through the Hilbert space filling curve. Intuitively, the hilbASR generalization algorithm partitions the data structure into blocks of k users: the first block from the user in position 0 to the user in position $k-1$ and so on (note that the last block can contain up to $2 \cdot k - 1$ users). The algorithm then returns the MBR computed considering the position of the users that are in the same block as the issuer. The worst case time complexity of hilbASR is $O(log(|I|))$.

We now present a new generalization algorithm, called *dicotomicArea*. Starting from the total area monitored by the LTS, Algorithm 1 iteratively calls a function, named *partitionArea*, that partitions the area into two adjacent rectangles of equal size. The partitioning is done along the horizontal and vertical axis depending from the variable *orient* having value HOR or VER, with these value being alternated at each iteration. The input of the algorithm consists of the degree of anonymity k and the issuer i. The output is **null** if less than k users are located in the total area monitored by the LTS, otherwise the algorithm returns an area in which at least k users are located. The al-

[1]For the purpose of this paper, there is no need to consider the "adaptive" data structure proposed in the paper.

gorithm terminates when at least 1 and at most $k-1$ users are located in any of the two sub-areas. Function $usersIn()$ is used to retrieve the set of users that are located in the area specified as parameter. Algorithm *dicotomicArea* is an instance of a class of algorithms presented in [1]; In that paper it is proved that any generalization algorithm that iteratively partitions the set of users, and that terminates when any block contains less then k users, is a Γ_I-safe algorithm. At each iteration, *dicotomicArea* partitions the set of users according to their location with respect to the sub-areas. If no user is located in a sub-area, then the set of users is not partitioned and iteration continues. On the contrary, if one of the sub-areas contains more than one user but less than k, execution terminates. The data structure that we used in the implementation of the algorithm is similar to the one we used for the implementation of *IntervalCloaking*. The only difference is that each internal node has two children instead of four. Consequently, the time complexity of the algorithm is the same as the one for *IntervalCloaking*.

Algorithm 1 DicotomicArea

```
1:  orient := HOR
2:  area := the total area
3:  if (|usersIn(area)| < k) then return null;
4:  while true do
5:    subAreas := partitionArea(area, orient)
6:    if ∃a ∈ subAreas s.t. 0 < |usersIn(a)| < k
      then
7:      return area
8:    else
9:      area := a_j ∈ subAreas s.t. i ∈ usersIn(a_j)
10:     if orient = HOR then orient := VER
11:     else orient := HOR
12:   end if
13: end while
```

A second generalization algorithm belonging to the class presented in [1] is called *dicotomicPoints*. The idea is to use a different partitioning function, named *partitionPoints*. The users are totally ordered according to their locations considering first one axis, then the other, and if necessary even the user identifier; Then, considering the user u in the middle[2], it partitions the users into two blocks: the ones before u, and the remaining ones Similarly to *dicotomicArea*, the *dicotomicPoints* algorithm alternates horizontal and vertical partitionings until any of the two blocks contain less than k users; Then it returns the MBR of all the users' locations in those two blocks. This algorithm has some similarities with the *Anonymize* algorithm presented in [4] despite they have been independently designed. The data structure that we used in the implementation of *dicotomicPoints* consists of two arrays, $order_x$ and $order_y$, containing the users ordered according to the horizontal and vertical axis, respectively. At each iteration, the user locations that are not in the same block as the issuer are removed from the two arrays. So, at each iteration it is necessary to find the user location in the middle of the correct array, to count how many users will be in each block and to remove the users that are not in the same block as the issuer. The first two operations can be performed in constant time, while the last one requires a time linear in the size of the two arrays. Since the number of iterations is logarithmic in the number of users and each iteration requires time linear in the number of the users, the worst case time complexity of the algorithm is $O(|I| \cdot \log(|I|))$.

[2]When there is an even number r of users, user u is the one in position r/2 +1.

3 Experimental results

We performed an extensive experimental evaluation of the algorithms presented in Section 2.We performed our tests using artificial data. Users' locations are randomly chosen and uniformly distributed[3] in a square-shaped area of side $10,000$. We identified two main parameters for each test: the value k, representing the cardinality of the anonymity set, and the total number p of users in the area. We are interested in three values in the output of the tests: a) the perimeter of the output region, b) the area of that region, c) the computation time. We implemented the algorithms using Java, and we performed our tests on a Linux machine with two 2,4Ghz Pentium Xeon processors and 4GB of shared RAM. All the output values presented in this section are obtained by running $1,000$ tests and taking the average or maximum value, as indicated in the specific experiment.

In order to compare the perimeter of the regions returned by the generalization algorithms with the one having the smallest perimeter, we implemented the *optimalUnsafe* generalization algorithm. This algorithm computes the set of $k-1$ users such that the perimeter of the MBR including these users and the issuer is minimal. The idea of *optimalUnsafe* is to search the best perimeter of the MBRs for each set containing the issuer and other $k-1$ users. Hence, the complexity of the algorithm is exponential in p; However, we developed several optimization techniques that make the algorithm in most cases computable in time lin-

[3]Despite uniform distribution may not be realistic, we believe it provides useful information on the general behavior of the algorithms.

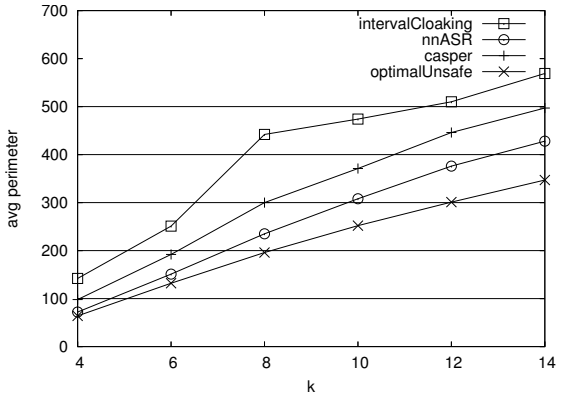

Figure 1. Average perimeter with $p = 50,000$.

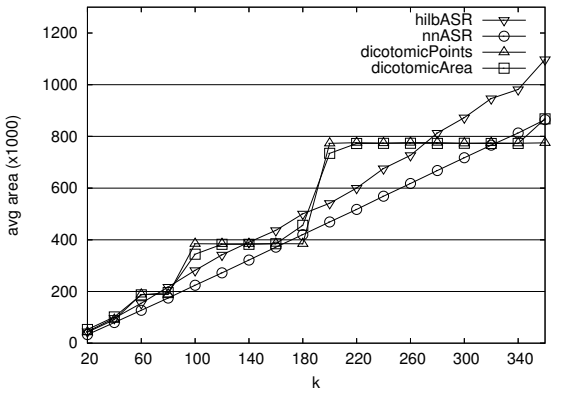

Figure 2. Average area with $p = 50,000$.

ear in the size of p, and exponential in the size of k. This makes it possible to compute the optimal perimeter, as a reference value for the evaluation of Γ_I-unsafe algorithms, for quite large values of p and practically relevant values of k.

Figure 1 shows the average perimeter of the region returned by four Γ_I-unsafe algorithms for different values of k. The principle behind the *nnASR* algorithm may induce the reader to think that the resulting region is minimal. Our empirical results show that that this is not the case. On average, *nnASR* returns regions having a perimeter 25% larger than the one of the region returned by *optimalUnsafe*. We also computed the average number of times in which *nnASR* returns the same result as *optimalUnsafe*. We noticed that this value rapidly decreases with the growing of k. For example, with $k = 4$ and $p = 50,000$, *nnASR* returns the region with the minimal perimeter in about 31% of the cases, while the percentage drops below 2% for $k = 14$ and the same number of users.

Figure 2 shows the average area of the regions returned by the algorithms *nnASR*, *hilbASR*, *dicotomicPoints* and *dicotomicArea* for different values of k. The considered algorithms show a similar behavior for values of k less than 50. While these values of k may be assumed sufficient for some applications, we consider the scalability of algorithms in terms of k very relevant, for example for their application in the enforcement of historical-k-anonymity [2]. Note, in Figure 2, that the derivates of the curves that refer to *hilbASR* and *nnASR* are almost constant while the derivates of the curves that refer to *dicotomicPoints* and *dicotomicArea* change significantly. This is due to the fact that *dicotomicPoints* partitions the number of points until it finds a set containing less than k users. The number of iterations is given by: $\lceil \log(\frac{p}{k}) \rceil$. Therefore, there are executions of the algorithm with different values of k that iterate the same number of times, hence computing, at the last iteration, the same number of users. Consequently, these executions return regions with similar area. An analogous argument holds for *dicotomicArea*.

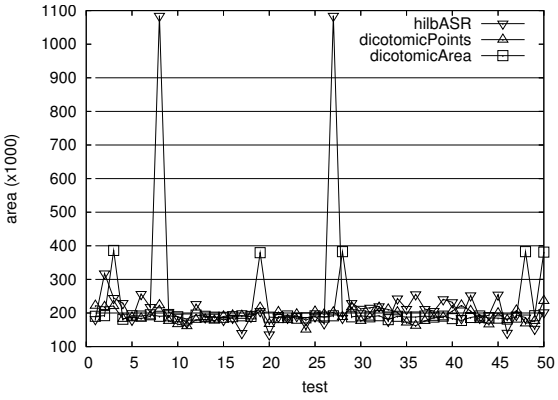

Figure 3. Area with $k = 80$ and $p = 50,000$.

Figure 2 also shows that, on average, *dicotomicPoints* and *dicotomicArea* return regions with the same area. This is due to the fact that, in our experiments, users are uniformly distributed, and hence the function *partitionArea* behaves similarly to *partitionPoints*.

Comparing *hilbASR*, *dicotomicPoints* and *dicotomicArea* we notice that, asymptotically, the average size of the regions returned by the three algorithms is the same. However, *hilbASR* has a more "regular" average growth with different values of k (Figure 2). The same holds by fixing k and changing the value of p.

On the other hand, we noticed that *hilbASR* has a less regular behavior when, fixing the values of k and p as well as the locations of all considered users, we vary the location from which the request is issued. In Figure 3, we report our preliminary experimental results about this phenomena. Each test consists in running

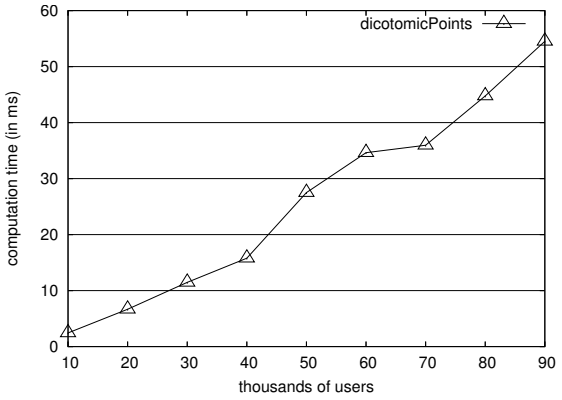

Figure 4. Computation time with $k = 80$.

the three algorithms with respect to an issuing location, randomly selected among the locations of the considered users. For the sake of readability, Figure 3 shows the result for 50 tests only, but we performed more than $1,000$ tests, also changing the fixed values of p, k, and users' locations. As we can observe in the figure, the size of the regions returned by the three algorithms is usually close to the average value (about $200,000$). However, in some cases, the execution of *dicotomicArea* results in an area that is almost double of the average one, while, in few cases, *hilbASR* results in an area that is almost 5 times larger than the average one. We do have some conjectures about the reason behind this observation, but a definite explanation requires further investigation.

Figure 4 shows the average computation time of the algorithms *dicotomicPoints* for different values of p. Average computation time of Algorithms *nnASR*, *dicotomicArea* and *hilbASR* is less than 1 ms in each experiment and we did not observe significant changes in the computation time for values of p between $10,000$ and $200,000$. This is due to the fact that i) the time complexity of the algorithms depends logarithmically in the size of p and ii) the computational time of the algorithms is dominated by startup time. On the contrary, the computation time of *dicotomicPoints* grows linearly with p. This result is consistent with the theoretical complexity analysis of the algorithm. We also evaluated the time complexity of the algorithms for a fixed p and different values of k and we observed that the execution time of the algorithms is almost not affected by the value of the parameter k.

According to our experiments, on average, *dicotomicPoints* and *dicotomicArea* return regions with similar size (in terms of area and perimeter). However, *dicotomicArea* can be computed more efficiently. We conjecture that *dicotomicPoints* would provide better results for non-uniform users' distributions.

4 Conclusion and Future Work

In this paper we presented a theoretical and empirical comparison of generalization algorithms. We also presented two new Γ_I-safe algorithms and we showed that, on average, the size of the area they return is similar to the size of the area returned by *hilbASR*, i.e., the only Γ_I-safe generalization algorithm previously proposed. The main difference is that the two new algorithms have a much smaller variance in the accuracy of their output when compared with *hilbASR*.

As future work, we intend to evaluate the performance of the algorithms in the case of non-uniform distributions of users, since we believe the algorithms may provide significantly different results. We are also tackling the problem of defining an optimal and Γ_I-safe generalization algorithm.

Acknowledgments

This work was partially supported by Italian MUR InterLink project N.II04C0EC1D. We would like to thank Dario Freni for his excellent programming job in the set-up of the experiments, and the reviewers for their helpful comments.

References

[1] C. Bettini, S. Mascetti, X. S. Wang, and S. Jajodia. Anonymity in location-based services: towards a general framework. In *Proc. of MDM*, 2007.

[2] C. Bettini, X. S. Wang, and S. Jajodia. Protecting Privacy Against Location-based Personal Identification. In *Proc. of SDM*, 2005.

[3] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of MobiSys*, 2003

[4] K. LeFevre, D. J. DeWitt and R. Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *Proc. of ICDE*, 2006.

[5] P. Kalnis, G. Ghinta, K. Mouratidis, and D. Papadias. Preserving anonymity in location based services. TRB6/06, National Univ. of Singapore, 2006.

[6] M.F. Mokbel, C. Chow, and W.G. Aref. The new casper: query processing for location services without compromising privacy. In *Proc. of VLDB*, 2006.