Journal of Location Based Services
Vol. 00, No. 00, Month 200x, 1–25

Spatial Generalization Algorithms for LBS Privacy Preservation *

Sergio Mascetti, Claudio Bettini, Dario Freni
DICo, University of Milan, Italy

X. Sean Wang
CS Dept, University of Vermont, USA

(*Received 00 Month 200x; In final form 00 Month 200x*)

Spatial generalization has been recently proposed as a technique for the anonymization of requests in location based services. This paper provides a formal characterization of a privacy attack that has been informally described in previous work, and presents a new generalization algorithm that is proved to be a safe defense against that attack. The paper also reports the results of an extensive experimental study, comparing the new algorithm with the ones that have been previously proposed in the literature.

1 Introduction

Location-based services (LBS) refer to those information services that deliver differentiated information based on the location from where a user issues the request. Thus, the user location information necessarily appears in a request sent to the service providers (SPs). A privacy problem arises in LBS when the user is concerned with the possibility that an attacker may connect the user's identity with the information contained in the service requests, including location and other information. In general, the association between the real identity of the user issuing an LBS request and the request itself as it reaches the service provider can be considered a privacy threat.

Previous works (3; 15; 16; 22) showed that simply dropping the issuer's personal identification data may not be sufficient to protect user's privacy. 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 narrow down the set of possible issuers to a small group. A notion of spatio-temporal k-anonymity was proposed by Gruteser et al. as a possible solution to guarantee user's privacy (15). The idea is to *generalize* the location (and time) information contained in a LBS request so that, based on that information, an attacker cannot deduce who is the issuer of the request among k potential issuers.

Example 1.1 Alice submits an LBS request from her device asking for a nearby vegetarian restaurant. She would rather prefer not to reveal that she is a vegetarian. For this purpose her credentials in the request to the service provider are substituted with a pseudo-ID and the network address of her device is also appropriately masked.

Suppose the *anonymized* request is obtained by an attacker. If this attacker happens to know that Alice was the only person to be present at the location and time indicated in the request (possibly through sighting, cameras, presence data in company buildings or the alike), the request is associated to her identity and Alice's private information is disclosed.

In order to prevent this privacy violation, Alice avoids providing her exact location and, instead, she sends to the service provider a *generalized area* that includes her location and the location of other $k-1$ users. The service provider replies to this *generalized request* with the set of vegetarian restaurants that are the closest one to any point of the generalized area.

If the generalized area is sufficiently small, the service result will probably consist of a small set of restaurants among which, there is the one that would have been returned if the exact location were provided. The advantage of using the generalized area is that, since the generalized area contains at least k users, even if the attacker can obtain Alice's request and even if he knows the location and the identity

*A preliminary version appeared as (20).

of each of the k users, he is not able to identify the issuer with probability greater than $1/k$ and therefore he is not able to infer that Alice is vegetarian (with probability greater than $1/k$).

This generalization technique implicitly assumes that the information about the spatio-temporal position of a sufficient number of potential users of the service is available to the entity performing the generalization. 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 the target service provider. The answer from the SP is also routed through the LTS to be redirected to the specific user with a refined result when possible (16).

In the above scenario, the generalization algorithm employed by the LTS has three goals: (a) to guarantee the user's privacy by ensuring that a sufficiently large number of potential users are not distinguishable from the issuer, (b) to preserve the quality of service by minimizing the size of the generalized area, and (c) to be efficient, since it must be computed on-line.

To illustrate the above three goals, consider a yellow pages LBS. If a user's request is not generalized, the SP will be able to provide a very accurate service pointing to the closest relevant resource considering the exact position of the user; However, by revealing that information, the user's privacy may be compromised. On the other hand, if the user's position is generalized to a very large area, many more resources would be selected as potentially relevant to the user; This could lead to a waste of computation time and bandwidth, as well as to the possible delivery of useless information to the user.

As mentioned above, a number of generalization solutions have been proposed in the literature. One of the goals of this paper is to define a common formal framework for the evaluation of these solutions. One of the main ideas of the framework is to explicitly define the assumptions on the knowledge available to the attacker as well as on his reasoning abilities. We note that the fact that certain generalization algorithms are considered "unsafe" by previous authors is simply due to different assumptions on the attacker's knowledge and reasoning abilities. We call "context" a set of these assumptions. The formalization allows us to formally prove the safety of generalization algorithms under different contexts.

In this paper we focus on contexts that have been implicitly assumed in most of previous work on this subject. More specifically, authors generally assume that the location of some users may be known to the attacker. Some authors also implicitly assume that the generalization function itself may be known to the attacker. They also usually limit the investigation of attacks to the derivation of private information from a single acquired request, assuming that no inferencing across multiple requests can be performed. Note that this assumption is nevertheless realistic for some practical application scenarios: on one side, there exist classes of services for which users would issue only occasional requests, making very hard any correlation between requests issued by the same user; on the other side, there are techniques (based for example on the use of different pseudo-IDs for each request from the same issuer) that can be effectively used to decrease the probability of correlation of requests even if they are close in time and space. We formalize these contexts in Section 3. Obviously, other assumptions are possible but are out of the scope of this paper.

A major effort has been devoted to the experimental evaluation of some of the generalization algorithms proposed in the literature as well as of a new algorithm proposed in this paper. While the algorithms can be proved or disproved to provide safe protection of privacy through the theoretical reasoning from the framework, they had to be evaluated in terms of the other two goals, namely, quality of service and efficiency.

The contributions of the paper may be summarized as follows: (i) we present a formal framework to model LBS privacy attacks and to evaluate the safety of defense techniques, based on an explicit representation of the adversary knowledge and reasoning capabilities (called *context* in the paper), (ii) we provide a theoretical and experimental evaluation of existing algorithms, (iii) we propose a new generalization algorithm and we show, through the experimental evaluation, that it is superior to existing solutions both in terms of the average area required for anonymization and for the variance of the same parameter.

The rest of the paper is organized as follows. In Section 2, we detail the generic LBS architecture that we are using for this paper. In Section 3, we formalize the necessary notions including contexts, attacks, and defenses, and then in Section 4, we present the existing generalization algorithms proposed in the literature

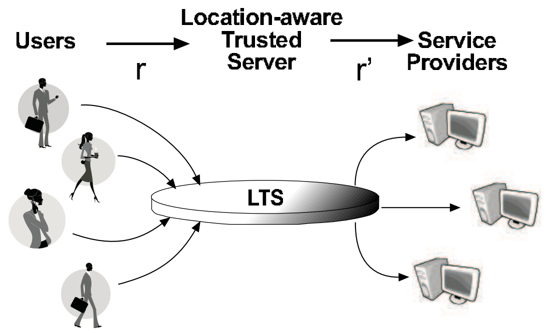


Figure 1. A general reference scenario

as well as new algorithms. In Section 5, we show the experimental results, in Section 6 we discuss related works and conclude with Section 7. In addition, the proofs of formal results are included in Appendix A.

2 Reference Scenario for LBS

Figure 1 shows our reference scenario that involves three entities: The **User** invokes or subscribes to location-based remote services that are going to be provided to her mobile device. The **Location-aware Trusted Server (LTS)** stores precise location data of all its users, using data directly provided by users' devices and/or acquired from the infrastructure. It also has the ability to efficiently perform spatio-temporal queries to determine, for example, which or how many users are in a certain region. The **Service Provider (SP)** fulfills user requests and communicates with the user through the LTS. Both pull and push communication service models are possible; We concentrate on the former but the framework we present can be easily extended to deal with the latter too.

Most of the approaches proposed in the literature (14; 15; 16; 22) to protect LBS privacy consider scenarios that can be easily mapped to the one depicted in Figure 1. Actually, scenarios where no location-aware intermediate entity is present have also been considered. For example, in (12) a direct communication between the user and the service provider is assumed, and the defense function is computed on the client system. Clearly in this model it is not possible to assume that the client has any awareness of the exact location of other clients; hence the generalization techniques proposed in this and in other papers would not be applicable. We believe that the current business models of mobile operators naturally support the existence and functionality of an entity like the LTS. Indeed, mobile users implicitly trust the operator infrastructure even if they know that very accurate information about their location and service requests is stored. Moreover, in most countries each operator has a very large number of customers, and hence a collection of location informations that may be more than sufficient to implement some of the defense techniques we are proposing.

The format of a request is represented by the following triple:

$$\langle IDdata, STdata, SSdata \rangle$$

- **IDdata** contains the exact user identity in the original request; when the request is generalized it is either empty or it contains a pseudo-id.
- **STdata** contains spatial-temporal information about the location of the user performing the requests, and the time the request was issued. For the sake of simplicity, we assume that this information is a point in 3-dimensional space (with time being the third dimension) for the original request, and a region in the same space for the generalized request. For the sake of simplifying notation, in the following of this paper we use **Sdata** and **Tdata** to denote the spatial and temporal part of $STdata$, respectively.
- **SSdata** contains parameters characterizing the required service and service provider.

3 A formal model for anonymity in LBS

3.1 *Formalization of attacks and defense functions*

Since attacks are performed on generalized requests, we start with defining generalization. As mentioned in the introduction, we focus in this paper on the technique in which the LTS uses a spatial *generalization* function to transform an original request into a generalized one and forwards it to the SP. To formally define such a function, we assume R is the set containing all the possible *original* requests issued by the users to the LTS as well as all the possible *generalized* requests that the LTS would forward to the SP. In the following we also use I to denote all the users that are potential issuers of requests, use $issuer(r)$ to denote the user who actually issued the request r, and indicate with loc_i (or $loc_i.x$, $loc_i.y$) the location of user i (or the location on the x, y axis, respectively).

Definition 3.1 Given the set R of requests, we say that $g : R \rightarrow R$ is a *generalization function.*

In the following, we often use r to denote an original request, and $r' = g(r)$ to denote the generalization of r that is forwarded to the SP. The generalization function is not necessarily a total function; if it happens to be undefined for some original request r, then the LTS simply suppresses r.

Most approaches to anonymity in LBS are based on generalization and/or suppression of requests. The main idea is to generalize the data that could be used, possibly joined with external knowledge, to re-identify individuals. The generalization should ensure that the re-identification process does not associate a request to its issuer with high enough probability. However, how much should we generalize in order to be safe? As observed in the introduction, most current approaches focus on $Sdata$ and devise a generalization function that is intended to provide safety even in the worst case scenario of an attacker having acquired external knowledge about the identities and locations of all the individuals in the spatio-temporal region specified in the request. This is a very conservative assumption. Nevertheless, we will show that the generalization function based on it is not sufficient if we assume, for example that the generalization function is known to the attacker.

We claim that the safety of a generalization function can only be formally evaluated if the re-identifying abilities of the attacker are clearly stated. To define such abilities, we introduce the concept of a *context*, denoted C, of a possible attack that consists of a set of assumptions:

- The assumptions about the explicit external knowledge that the attacker could obtain; this could be a public knowledge (e.g., a voter list) or confidential information (e.g., the identity of a user in a given location).
- The assumptions about the reasoning abilities of the attacker; For example, the assumption about the fact that the attacker is (or is not) able to reason with multiple requests issued by the same user in different time instants.

Clearly, C changes depending on the applicative context and on the level of conservativeness of the chosen model. In this paper we assume for all considered contexts that the attacker has basic reasoning abilities like, for example, the ability to check for spatial inclusion. We also assume, as public knowledge, that each generalization function is such that the generalized region includes the location of the issuer. That is, the LTS does not lie about the location information of the issuer, but only fuzzifies it. Formally, for each generalization function g and for each original request r, $r.Sdata \in g(r).Sdata$. In Sections 3.2 and 3.3 we show the two more common scenarios addressed in the literature, each one modeling different context assumptions.

Once the context assumptions are defined, we can formalize how the attacker can try to infer, from a generalized request, the identity of the user that issued it. We model this attack as the likelihood of associating a specific identity to a generalized request. Definition 3.2 formally states what is an attack by normalizing to one all the likelihood values for a given generalized request.

Definition 3.2 An *attack* based on context assumptions C is a function $Att_C : R \times I \rightarrow [0, 1]$ such that for each generalized request r',

$$\sum_{i \in I} Att_C(r', i) = 1 \tag{1}$$

Given an attack Att_C and a generalized request r', we indicate with AS_C the function that associates to each generalized request r' the *Anonymity Set of* r' i.e., the set of users that have a non zero probability of being identified as the issuers of r'. Formally, $AS_C(r')$ is the set of all users $i \in I$ such that $Att_C(r', i) > 0$.

By Definition 3.2, attacks can be specified in which, given a request r', the users in the anonymity set of r' have different probabilities of being the issuer of r', as shown in Example 3.3.

Example 3.3 Consider a location based yellow pages service and the following context C:

- attacker could acquire the knowledge of the location of each user;
- attacker could acquire the knowledge of the users' profiles (possibly obtained during the registration phase);
- the attacker is not able to reason with multiple requests

Suppose that Alice issues a request asking for the closest shop where she can find classical music. The LTS receives the request and deletes the information that could directly lead to Alice's identity (her name, for example). Moreover, the exact location of Alice is generalized into an area. Then, the resulting generalized request r' is forwarded to the SP.

If an attacker obtains r', he first uses the location knowledge to restrict the set of possible issuers to the users whose location is in the region specified in r'. Suppose this set is composed by three users: Alice, Bob and Carl. Then, the attacker uses the knowledge of the users' profiles, to discover that *Alice*'s musical interests are closer to the classical genre than those of *Bob* and *Carl*. Hence, this attack is characterized by a higher likelihood for Alice being the issuer of r', than *Bob* or *Carl*.

A special case of the general definition of attack is the one in which all the users in the anonymity set have the same probability of being the issuer. We call these attacks *uniform*.

Definition 3.4 An attack Att_C is *uniform* if, for all generalized request r', for each pair of users $i, i' \in AS_C(r')$, $Att_C(i, r') = Att_C(i', r')$.

The relationship between a uniform attack and the anonymity set is formalized by the following proposition.

PROPOSITION 3.5 *Given a uniform attack Att_C based on context C, for each generalized request $r' \in R$ and for each $i \in I$,*

$$Att_C(r', i) = \begin{cases} 0 & \textit{if } i \notin AS_C(r') \\ \frac{1}{|AS_C(r')|} & \textit{otherwise} \end{cases}$$

For the sake of simplicity, in the remainder of this paper, when describing a specific uniform attack, we will simply identify the corresponding anonymity set, since it completely characterizes the attack. In the following we also indicate with $UAtt_C$ the uniform attack characterized by AS_C.

Clearly, an attack Att_C depends on the context assumptions C: the more information the attacker has available and the more his reasoning abilities the higher the probability of identifying the issuer.

Given the definition of attack, we define when a generalized request r' can lead the attacker to infer the correct identity of the issuer of r'. The idea of Definition 3.6 is that a request is safe if the attack associates it to the correct identity with a likelihood smaller than a threshold value h.

Definition 3.6 Let Att_C be an attack, h a value in $[0, 1)$ and r' a generalized request. We say that r' is a *safe request against Att_C with threshold h* if, given $i = issuer(r)$, $Att_C(r', i) \leq h$.

If a request is not safe, we say that it is *unsafe*.

The task of the LTS is to avoid to generalize a request into a unsafe one. We call *defense function* a generalization function that generates only requests that are safe against a given attack.

Definition 3.7 Let Att_C be an attack and h a value in $[0, 1)$. A generalization function $g : R \to R$ is a *defense function against* Att_C *with threshold* h if for each original request $r \in R$ such that $g(r)$ is defined, $g(r)$ is a safe request against Att_C with threshold h.

3.2 *Modeling spatio-temporal anonymity*

Many papers in this research area explicitly or implicitly considered a common context that we refer to as C_{st} and that we model in this section (14; 15; 16; 22).

The idea of the C_{st} context is that users can be identified through their location, and therefore user's privacy is endangered if a generalized request contains precise information about issuer's location. For example, an attacker can infer user's identity from user's location through the physical observation of the user, or because the user is the only one that can be in that location (a suburban house, for instance).

In context C_{st} it is assumed that (i) the attacker could acquire the knowledge about the exact location of each user, (ii) the attacker knows that the region $g(r).Sdata$ always includes the point $r.Sdata$ and (iii) the attacker is not able to reason with more than one request. It may seem unrealistic that the attacker knows the location of each user; however, if an attacker can possibly know the location of one user (not too outrageous an assumption), it is quite natural to assume the worst case, namely he knows the location of all users. This assumption may be relaxed by saying that the attacker can only know the locations of some users in some particular areas. Such a partial location assumption is formalized in (5) but will not be used in this paper.

The uniform attack based on context C_{st} is characterized by the anonymity set of users who are located within the area specified in the generalized request.

Definition 3.8 Let C_{st} be the context in which it is assumed that for each identity $i \in I$ the attacker knows the association of i with the location loc_i.

For each generalized request r', the anonymity set based on context C_{st} is:

$$AS_{C_{st}}(r') = \{i \in I | loc_i \in r'.Sdata\}$$

The idea is that the more users are located inside $r'.Sdata$, the more anonymous is the issuer of r'.

Definition 3.9 In context C_{st}, a generalized request r' is said to be *k-anonymous* if $|AS_{C_{st}}(r')| \geq k$.

The solutions to the anonymity problem in context C_{st} provided in (14; 15; 16; 22) guarantee anonymity by generalizing the *Sdata* field of the incoming request and forwarding to the SP only requests r' such that r' is k-anonymous in context C_{st}.

Theorem 3.10 shows that, considering context C_{st}, a generalization function that generates k-anonymous requests is a defense function, and that the parameter k is the inverse of h. The proof of the theorem can be found in Appendix A.

THEOREM 3.10 *Let k be an integer and $g()$ a generalization function such that, for each original request r, $g(r)$ is k-anonymous in context C_{st}. Then, $g()$ is a defense function against $UAtt_{C_{st}}$ with threshold $1/k$.*

3.3 *Modeling spatio-temporal anonymity when the generalization function is known to the attacker*

In this section we give a formal explanation of the attacks illustrated in the literature against the defense techniques first proposed in context C_{st}. In a nutshell, the attacks implicitly consider the generalization function as publicly known, while the defense technique assumed otherwise.

Assume an attacker obtains a generalized request r' and knows, in addition to the knowledge of C_{st}, the generalization function g used by the LTS. In this case, the attacker can compute, for each user $i \in I$ the *potential original request* r_i that would be send to the LTS if i was the issuer. Then the attacker can check if $g(r_i)$ equals r'. If it does, then i is a potential issuer, otherwise the attacker can understand that i is not the issuer of r'.

Example 3.11 User i_1 issues a request r_1 to the LTS. The required anonymity threshold h is 1/3, hence r_1 is generalized, using algorithm g, to a request r' such that i_1, i_2 and i_3 are located inside $r'.Sdata$.

If an attacker obtains r', he can compute, for each user $i \in I$ the potential original request r_i and then apply g to each r_i. Since $r'.Sdata$ includes the locations of i_1, i_2 and i_3 only, for any user i not in $\{i_1, i_2, i_3\}$ the potential request r_i is generalized to a request $r'' \neq r'$. So, the only three users whose potential request could be generalized to r' are i_1, i_2 or i_3. Assume that r_3 is generalized to a request r''' that is different from r' while both r_1 and r_2 are generalized into r'. Figure 2 gives a graphical representation of this situation.

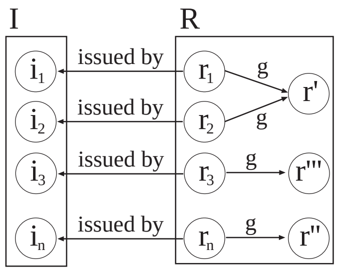


Figure 2. Example of attack in context C_{st+g}.

Since i_1 and i_2 have the same probability of being the issuer of r', the attacker can identify the issuer of r' with probability 1/2, hence r' is unsafe with respect to this attack with the required threshold of 1/3.

We indicate with C_{st+g} the context in which the attacker, in addition to the knowledge and reasoning abilities of the context C_{st}, also has the knowledge of the function $g()$. As depicted in Example 3.11, in this context a possible uniform attack can be based on the following definition of anonymity set:

Definition 3.12 Let C_{st+g} be the context C_{st} in which the attacker also knows: $\{\forall r \in R,\, g(r)\}$.

For each generalized request r', we define the attack $UAtt_{C_{st+g}}$ as the uniform attack characterized by the anonymity set $AS_{C_{st+g}}(r') = \{i \in I | \exists r \in R \ s.t. \ loc_i \in r'.Sdata, issuer(r) = i \ and \ g(r) = r'\}$.

Intuitively, Definition 3.12 states that the anonymity set of a request r' based on context C_{st+g} is composed of all users $i \in I$ such that if i were the issuer of the original request, then the generalization function $g()$ would return the same r'.

Theorem 3.13 shows that the uniform attack in context C_{st+g} has more reidentification power than the uniform attack in context C_{st}.

Theorem 3.13 *For each generalized request* r', $AS_{C_{st+g}}(r') \subseteq AS_{C_{st}}(r')$.

Intuitively, Theorem 3.13 states that, given a generalized request $r' = g(r)$, the set of possible issuers of r' computed exploiting the knowledge of the function $g()$ is a subset of the set of users whose location is within $r'.Sdata$. This implies that, if $g()$ is known to the attacker, a request that is k-anonymous according to Definition 3.9, may nevertheless be associated to the issuer with likelihood greater than $1/k$.

Corollary 3.14 *Let* r' *be a generalized request. The* k*-anonymity of* r' *in context* C_{st} *is a necessary but not sufficient condition for* r' *to be a safe request against* $UAtt_{C_{st+g}}$ *with threshold* $1/k$.

The "necessary" part immediately follows Theorem 3.13, while the "not sufficient" part is shown with Example 3.15.

Corollary 3.14 shows that the notion of k-anonymity of a request is not sufficient in order to guarantee user's privacy when the generalization function is known to the attacker. In Example 3.15 we show, in terms of our framework, why the generalization function proposed in (15) is not a defense function against $UAtt_{C_{st+g}}$ with threshold $1/k$.

Example 3.15 Figure 3 shows four user locations. Users u_1, u_2 and u_3 are close to each other, while u_4 is far away. Consider a generalization function $g()$ that returns a request having, as generalized location, the minimum bounding rectangle that includes the location of the issuer as well as the locations of the $k-1$ users nearest to the issuer. In our example, a request issued by u_1, u_2 or u_3 is generalized, with degree of anonymity equal to 3, into a request that has A_2 as $Sdata$ while a request issued by u_4 with the same degree of anonymity is generalized into a request that has A_1 as $Sdata$.

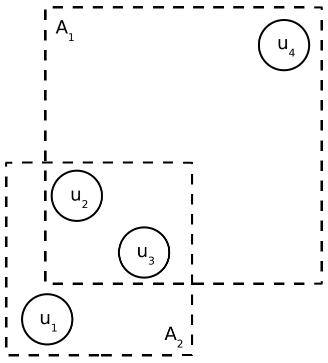


Figure 3. Example of attack when the generalization function is known to the attacker.

As we proved in Theorem 3.10, $g()$ is a defense function against the uniform attack characterized by $AS_{C_{st}}$ with threshold 1/3. On the contrary, if we assume that the attacker knows the generalization function, then $g()$ is not a defense function against the uniform attack $UAtt_{C_{st+g}}$ for any threshold in $[0, 1)$. Indeed, let r be a request issued by u_4 and r' the generalization of r such that $r'.Sdata = A_1$. Since any request issued by u_1, u_2 or u_3 would be generalized into a request different from r', the only possible issuer of r' is u_4.

In the following of this paper, we indicate with C_{st+g}*-unsafe* the generalization algorithms that compute a defense function against $UAtt_{C_{st}}$ with a given threshold h but that are not a defense function against $UAtt_{C_{st+g}}$ with threshold h. On the contrary, we indicate with C_{st+g}*-safe* the generalization algorithms that compute a defense function against $UAtt_{C_{st+g}}$ with threshold h (and consequently compute a defense function against $UAtt_{C_{st}}$ with threshold h).

3.4 *Usage of the framework to define a defense function*

We show how the formal framework defined in Section 3 can be useful to define a defense function against a given attack. As an example, we consider $UAtt_{C_{st+g}}$, as defined in Section 3.3, and we show how to define an algorithm to compute a defense function. In the remainder of the paper we always assume a given set of users I considered in all definitions, generalization algorithms, and formal results, with the property of having cardinality larger than the anonymity parameter k. When C_{st} and C_{st+g} are considered it is also assumed that the generalization algorithms can use the information about the location of each user in I.

Since the attacker's knowledge assumed in C_{st+g} subsumes the attacker's knowledge in C_{st}, and in the two contexts the attacker has the same reasoning abilities, the defense function against $UAtt_{C_{st+g}}$ should also provide protection against $UAtt_{C_{st}}$. This consideration suggests us a two steps procedure to define the algorithm that computes the defense function against $UAtt_{C_{st+g}}$:

(i) define algorithm Gen_{st} that provides protection against $UAtt_{C_{st}}$;
(ii) alter Gen_{st} to obtain Gen_{st+g} that provides protection against $UAtt_{C_{st+g}}$.

In Algorithm *1*, we define the Gen_{st} algorithm inspired by the *IntervalCloaking* generalization algorithm (15) that will be described in details in Section 4.1. The idea of Gen_{st} is to iteratively restrict the anonymity set that is stored in the variable AS initialized to I. At each iteration AS is partitioned. If the block b that contains the issuer contains at least k users, then AS is set to b and the iteration continues. Otherwise, algorithm terminates returning the set of users in AS. The generalized location is computed as the MBR of the locations of the users in this set at the time r is issued. Then, the final generalized request r' is obtained substituting the generalized location to the exact location, and dropping information in $IDdata$ that could explicitly identify the issuer (possibly substituting it with a pseudo-id). Here and in the following, these steps are omitted from the algorithms' description for simplifying the presentation.

Algorithm *1* Gen_{st}

- **Input**: an original request r, a positive integer value k.
- **Output**: a set of users with cardinality at least k that contains $issuer(r)$.
- **Method**:

```
1: $AS := I$;
2: $cont$ :=true
3: while ($cont$) do
4:    partition $AS$ into $AS_1, \ldots, AS_n$;
5:    $b := AS_j$ where $AS_j$ is the block in $AS_1, \ldots, AS_n$ that contains $issuer(r)$
6:    if ($|b| < k$) then
7:       $cont$ :=false;
8:    else
9:       $AS := b$
10:   end if
11: end while
12: return $AS$
```

Gen_{st} does not automatically provide protection against $UAtt_{C_{st+g}}$ due to a problem similar to that described in Example 3.15. A solution consists in: i) imposing the partitioning function to be independent from the issuer's location and ii) modifying the termination condition of Gen_{st} so that the algorithm terminates if any of the blocks into which AS is partitioned contains less than k users. The resulting algorithm is Gen_{st+g} and is shown as Algorithm *2*.

Algorithm Gen_{st+g} iteratively restricts the set AS of candidate identities by successively partitioning it. The iteration terminates when a further partitioning would contain at least one block with less than k users. Then, the algorithm returns the set of the users in the anonymity set.

Algorithm *2* Gen_{st+g}

- **Input**: an original request r, a positive integer value k.
- **Output**: a set of users with cardinality at least k that contains $issuer(r)$.
- **Method**:

```
1: $AS := I$;
2: $cont$ :=true
3: while ($cont$) do
4:    partition $AS$ into $AS_1, \ldots, AS_n$;
5:    if ($\exists j \in \{1, \ldots, n\}$ such that $|AS_j| < k$) then
6:       $cont$ :=false;
7:    else
8:       $AS := AS_j$ where $AS_j$ is the block in $AS_1, \ldots, AS_n$ that contains $issuer(r)$;
9:    end if
10: end while
11: return $AS$
```

Note that Algorithms *1* and *2* do not specify how the users should be partitioned during each iteration.

Hence, in practice, Algorithms *1* and *2* are classes of algorithms that can be instantiated providing a specific procedure to compute the partition. For example, the aforementioned *IntervalCloaking* generalization algorithm (15) is an instance of the class Gen_{st}. In Section 4.2 we will present some algorithms that are instances of the class Gen_{st+g}.

The correctness of Algorithm *2* is given by the fact that the set of users returned by $Gen_{st+g}(r, k)$ is such that, if any of them issues a request r_1, $Gen_{st+g}(r_1, k)$ would return the same set of users. In order to have this property, the partitioning procedure should be computed independently from r. This is the intuition of the proof of Theorem 3.16 that formalizes the correctness of the algorithm.

Theorem 3.16 *Let an integer* $k \in \mathbb{Z}^+$ *be the input parameter of algorithm* Gen_{st+g}. *If the partition procedure of Line 4 of Algorithm 2 is deterministic and has AS as the only parameter, then algorithm* Gen_{st+g} *computes a defense function against* $UAtt_{C_{st+g}}$ *with threshold* $1/k$.

4 Algorithms for spatial generalization

In this section, we outline the generalization algorithms that were previously proposed in the literature, including one recently proposed by us, and we present a new algorithm. For each algorithm, we discuss the worst-case time complexity; experimental results are then presented in Section 5.

In Section 4.1 we present the C_{st+g}-unsafe generalization algorithms; for the sake of brevity, we do not demonstrate, for each algorithm, that it does not compute a defense function against $UAtt_{C_{st+g}}$. Indeed, a counterexample like the one illustrated in Example 3.15 can be easily found. On the contrary, in Section 4.2, we present the C_{st+g}-safe algorithms and we formally prove that they compute a defense function against $UAtt_{C_{st+g}}$.

4.1 C_{st+g}*-unsafe generalization algorithms*

The first generalization algorithm that appeared in the literature was named *IntervalCloaking* (15). 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, hence the algorithm has a time complexity of $O(|I|)$. However, in the special case of users uniformly distributed in the considered area, the height of the tree is logarithmic in the number of users, and hence in this special case the algorithm has a complexity of $O(\log(|I|))$.

Casper is a framework for privacy protection that includes a generalization algorithm (22). 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;

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

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 Neighbour query among the users and return the MBR of the result. Following this idea, the *nnASR* generalization algorithm is proposed (16); the strategy of this algorithm is picking a random user i in the set of the $k-1$ users that are closest to the issuer, and returning 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 Neighbour queries in logarithmic expected time with respect to the number of users.

4.2 C_{st+g}-*safe generalization algorithms*

To the best of our knowledge, the first C_{st+g}-safe generalization algorithm that was proposed is called *hilbASR* (16). The algorithm is an instance of Gen_{st+g} in which the partitioning is obtained exploiting the Hilbert space filling curve[1] 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 block that contains the issuer. The worst case time complexity of hilbASR is $O(log(|I|))$. Indeed, the algorithm does not actually compute all the blocks of the partition, but instead only finds the block that contains the issuer. Therefore, the only non-constant operation that is required is to find the issuer in the data structure, and this operation can be completed in logarithmic time.

It can be easily seen that *hilbASR* is an instance of Gen_{st+g} in which only one iteration is performed.

THEOREM 4.1 *If integer* $k \in \mathbb{Z}^+$ *is given as input parameter of algorithm* hilbASR*, then the algorithm computes a defense function against* $UAtt_{C_{st+g}}$ *with threshold* $1/k$.

In the preliminary version of this paper (20), we proposed a C_{st+g}-safe generalization function called *dichotomicPoints* that is shown in Algorithm *3*. The algorithm iteratively partitions the set of users into two sets. The idea is that the users are totally ordered according to their locations considering first one axis, then the other, and if necessary even the user identifier[2]; Then, considering the user u whose position is in the middle of the totally ordered set of positions[3], the algorithm partitions the users into two blocks: the ones before u, and the remaining ones. Iteration continues considering only the users in the block that contains the issuer.

At each iteration, in order to choose which axis to consider first for the total ordering, the algorithm computes the maximum and minimum values of users' locations projected on each axis. Then, the axis having the maximum value of the difference is chosen (Lines 3 to 13). The *dichotomicPoints* algorithm terminates when the block that contains the issuer has cardinality smaller than $2 \cdot k$; Then, it returns the set of the users in the block.

We should mention that the idea of computing a generalization by iteratively splitting the users along one of the dimensions also appeared in the *Anonymize* algorithm presented in (13) to generalize database relations.

Example 4.2 Consider the positions of 20 users, as shown in Figure 4(a). Assume that user i issues a request that should be made 2-anonymous. The *dichotomicPoints* algorithm computes the necessary

[1] A space filling curve transforms 2-D coordinates into 1-D coordinate. With high probability, two points that are close to each other in the 2-D coordinates are also close to each other in the transformed 1-D coordinate.

[2] We assume that each user has a unique user identifier.

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

Algorithm *3 dichotomicPoints*

- **Input**: an original request r, a positive integer value k.
- **Output**: a set of users with cardinality at least k that contains $issuer(r)$.
- **Method**:

```
1: the array $AS$ is initialized with the identities of all users ($I$)
2: while ($|AS| \geq 2 \cdot k$) do
3:   $min_x := min_{i \in AS}(loc(i).x)$
4:   $max_x := max_{i \in AS}(loc(i).x)$
5:   $min_y := min_{i \in AS}(loc(i).y)$
6:   $max_y := max_{i \in AS}(loc(i).y)$
7:   if $((max_x - min_x) \geq (max_y - min_y))$ then
8:     $firstOrder := x$
9:     $secondOrder := y$
10:  else
11:    $firstOrder := y$
12:    $secondOrder := x$
13:  end if
14:  sort $AS$ according first to $firstOrder$, then to $secondOrder$, and eventually to user identifiers.
15:  $pivot := \lfloor |AS|/2 \rfloor$
16:  $issuerIndex :=$ the index such that $AS[issuerIndex] = issuer(r)$
17:  if ($issuerIndex < pivot$) then
18:    $AS := AS[0] \ldots AS[pivot - 1]$
19:  else
20:    $AS := AS[pivot] \ldots AS[|AS| - 1]$
21:  end if
22: end while
23: return the set of user identities in $AS$
```

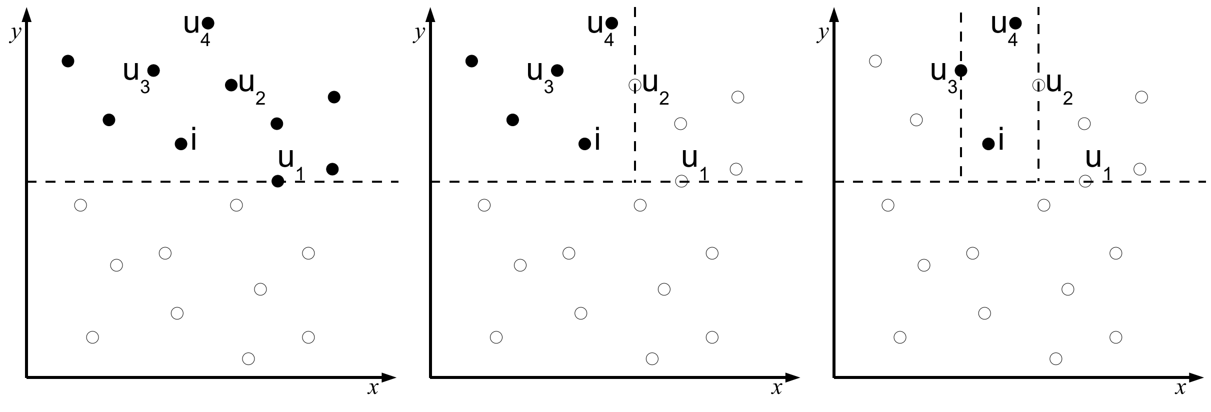


Figure 4. Example of *dichotomicPoints* algorithm

generalization in three iterations of its main cycle. Each of the three Subfigures 4(a) 4(b), and 4(c) shows the set of positions included in the current AS array computed by that iteration (as black filled circles), as well as the rest of the users' positions.

The array AS is reassigned to contain the identities of the 20 users in our example. In the first iteration, the difference between the maximum and minimum value over the set of all projections of user positions on the x axis is compared with the difference between the maximum and minimum value on the y axis. Since in this example the difference along the y axis is larger, the array AS is sorted considering the values on this dimension first. The *pivot* variable is assigned with the value 10, which in AS happens to be the index of user u_1. Since the issuer i has index 12, AS is reassigned to contain only the identities of u_1 and of the users with greater indexes (i.e., $AS := AS[10], \ldots, AS[19]$).

In the second iteration, considering the projections on the x and y axis of the positions of the 10 users currently in AS, the x axis is selected as $firstOrder$; the reason can be clearly seen observing the black filled circles in Figure 4(a) that also represents the set of positions considered in the second iteration. Hence, this time AS is sorted considering first projections of positions on x. In the resulting array, the *pivot* is computed as 5, which in Figure 4(b) corresponds to user u_2. Since now the issuer i has index 3, AS is reassigned to contain only the identities of the users with indexes smaller than 5, i.e., the users located at the left of u_2. ($AS := AS[0], \ldots, AS[4]$).

In the third iteration, AS is again sorted considering first projections of positions on x. The *pivot* is 2, which in Figure 4(c) corresponds to user u_3. Since the issuer i has index 3, AS is reassigned to contain $AS[2]$, $AS[3]$, and $AS[4]$, i.e., the identities of the users with indexes greater than or equal to 2, discarding the users whose location is at the left of u_3.

Since AS now contains less than $2 \cdot k = 4$ users, the algorithm terminates returning users i, u_3 and u_4.

Note that *dichotomicPoints* terminates when a block contains less then $2 \cdot k$ users and therefore any further partitioning would generate a block with less than k users. Therefore, *dichotomicPoints* is an instance of the class of generalization functions presented in Section 3.4, hence it is a defense function against $UAtt_{st+g}$.

THEOREM 4.3 *If integer $k \in \mathbb{Z}^+$ is given as input parameter of algorithm* dichotomicPoints*, then the algorithm computes a defense function against $UAtt_{C_{st+g}}$ with threshold $1/k$.*

The data structure that we used in the implementation of *dichotomicPoints* 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 $O(|I| \cdot \log(|I|))$.

We now present a new C_{st+g}-safe generalization algorithm that is called *grid*. This algorithm partitions the set of users in two steps. During the first step, users are totally ordered considering their location along the x axis, then along the y axis and eventually according to their user identifier. This ordered set of users is then partitioned into blocks of consecutive users, each block having the same number of users except the last block that may contain more users than the previous ones. Only the users that are in the same block as the issuer, are considered in the second step in which users are ordered considering first their location along the y axis, then their location along the x axis and eventually according to their user identifier. Using this ordering, users are partitioned similarly to what is done in the first step. Eventually, the set of users that are in the same block as the issuer is returned.

Example 4.4 Figure 5(a) shows the same position of 20 users as considered in Example 4.2 with the issuer i of a request in the same position. We show the execution of the *grid* algorithm, in the case 2-anonymity is required.

The array AS is set to contain the identities of the 20 users. In the first step, AS is sorted considering first the values of users' locations on the x axis. Since the number of blocks (*nob*) into which AS should be partitioned is $\left\lfloor \sqrt{20/2} \right\rfloor = 3$, the number of users in each block (*upb*) is computed as $\lfloor 20/3 \rfloor = 6$. Then, since the issuer has index 8 in AS, the index of the block containing the issuer (*ibi*) is computed as $\lfloor 8/6 \rfloor = 1$. Consequently, AS is reassigned to the set of users in the same block as the issuer. More technically, *start* and *end* are computed as 6 and 11, and AS is set to contain only $AS[6], \ldots, AS[11]$.

In the second step, AS, currently containing 6 users (identified in Figure 5(b) as black filled circles) is sorted considering first the users' positions along the y axis. Since the value of *nob* cannot change after it is initialized, AS is again partitioned into 3 blocks. Considering that AS now contains 6 user identities,

Algorithm 4 *Grid*

- **Input**: an original request r, a positive integer value k.
- **Output**: a set of users with cardinality at least k that contains $issuer(r)$.
- **Method**:

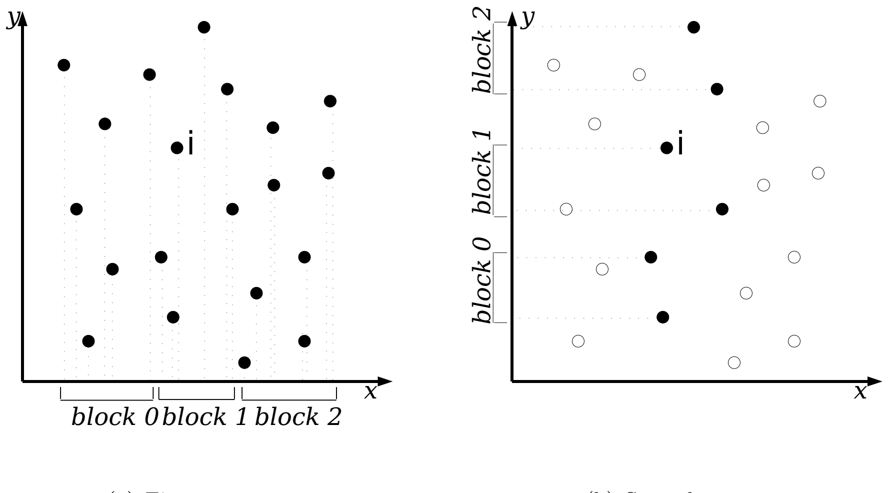
```
1: the array $AS$ is initialized with the identities of all users ($I$)
2: $nob := \left\lfloor \sqrt{|AS|/k} \right\rfloor$ {number of blocks}
3: if ($nob \leq 1$) return the set of all user identities ($I$)
4: for ($dim \in \{x, y\}$) do
5:   if ($dim = x$) then
6:     sort $AS$ according first to the location on $x$, then to the location on $y$ and eventually to the user identifier.
7:   else
8:     sort $AS$ according first to the location on $y$, then to the location on $x$ and eventually to the user identifier.
9:   end if
10:  $issuerIndex$ := the index such that $AS[issuerIndex] = issuer(r)$
11:  $upb := \lfloor |AS|/nob \rfloor$ {users per block}
12:  $ibi := \lfloor issuerIndex/upb \rfloor$ {issuer's block index}
13:  if ($|AS| \bmod nob = 0$ OR $ibi < nob - 1$) then
14:    $start := ibi \cdot upb$
15:    $end := start + upb - 1$
16:  else
17:    $start := (nob - 1) \cdot upb$
18:    $end := |AS| - 1$
19:  end if
20:  $AS := AS[start] \ldots AS[end]$
21: end for
22: if ($|AS| \geq k$) then
23:   return the set of user identities in $AS$
24: else
25:   return null
26: end if
```

each of the new blocks has to contain 2 users. Since the issuer has now index 3 in AS, the index of the block that contains the issuer (ibi) is computed as $\lfloor 3/2 \rfloor = 1$. Therefore, AS is reassigned to contain only $AS[2]$ and $AS[3]$, and this is the set of users' identities returned by the algorithm.

Theorem 4.5 proves that the *grid* algorithm is C_{st+g}-safe. Intuitively, the correctness of the algorithm relies on the fact that the product of the number of blocks into which users are partitioned along the two dimensions is less than or equal to $\lfloor |I|/k \rfloor$. This implies that the set I of users is partitioned at most into $\lfloor |I|/k \rfloor$ blocks of about the same size, hence each block contains at least k users. This is the reason why the number of blocks along each dimensions is $\left\lfloor \sqrt{|I|/k} \right\rfloor$.

Theorem 4.5 *If integer $k \in \mathbb{Z}^+$ is given as input parameter of algorithm* grid*, the algorithm computes a defense function against $UAtt_{C_{st+g}}$ with threshold $1/k$.*

In order to compute the *grid* algorithm, we use a data structure that keeps the users totally ordered according to their location along the x axis, then along the y axis and eventually according to their user identifier. Therefore, during the first step of the algorithm it is not necessary to sort the set of users and the only non-constant operation is to find the index of the issuer. This operation has a worst case time complexity logarithmic in the size of $|I|$. During the second step, it is necessary to sort the users that, during the first iteration were in the same bucket as the issuer. Since these users are at most $2(|I|/\left\lfloor \sqrt{|I|/k} \right\rfloor) - 1$,



(a) First step (b) Second step

Figure 5. Example of *grid* algorithm

in the worst case this operation can be performed in time $O(\sqrt{k|I|} \cdot \log \sqrt{k|I|})$. Hence, the worst case time complexity of *grid* algorithm is given by $O(\log |I| + \sqrt{k|I|} \cdot \log \sqrt{k|I|})$ that, since $|I|$ and k are positive integers, is equal to $O(\sqrt{k|I|} \cdot \log \sqrt{k|I|})$.

As we will show in Section 5 the computation time of the algorithm increases linearly with k and I. In order to lower the computation time of the algorithm and improve its scalability, instead of maintaining a single data structure that orders all the users according to the x axis, we partition I and maintain the same data structure for each block. Then, the complexity of the *grid* algorithm is $O(\sqrt{k|B|} \cdot \log \sqrt{k|B|})$ where B is the block that contains the issuer. This partitioning is used as a preprocessing step for the *grid* algorithm.

In our experiments, we partitioned the users according to their locations. The idea is that the total area monitored by the LTS is partitioned into a grid having the same number of rows and columns. One issue with this approach is that if the block from which a request was issued contains less than k users, the generalization fails. Therefore, the number of rows and colums of the grid must be determined considering the average density of users in the area, making sure that the number of users in each block is larger than k. If these conditions hold, this preprocessing technique preserves the safety of the generalization algorithm.

An important feature of the *grid* algorithm is that it can be easily extended to consider an arbitrary number of dimensions, making it possible to finely tune the importance of each of them. This characteristic can be used, for example, to consider speed and direction in a context in which the attacker is able to reason with traces of requests. If an arbitrary number d of dimensions is used, each of them should be partitioned into $\alpha = \left\lfloor \sqrt[d]{\lfloor |I|/k \rfloor} \right\rfloor$ blocks since, similarly to the bidimensional case, the product of the number of blocks in each dimension should be less then or equal to $\lfloor |I|/k \rfloor$.

In general, all the C_{st+g}-safe algorithms can be easily extended to compute the generalizations along an arbitrary number of dimensions. However, it is not clear to us how *hilbASR* or *dichotomicPoints* may be easily adapted to specify the relevance of each dimension. On the contrary, using *grid*, it is possible to associate a weight w_i to each dimension d_i. A weight affects the relevance of a dimension by influencing the number of blocks into which the set of users is partitioned along that dimension. The higher a dimension's weight, the more the blocks. Intuitively, given that the weights are such that their product is equal to one (if not, they can be easily rebalanced), the number of blocks along a dimension d_i with weight w_i is equal to $\lfloor w_i \cdot \alpha \rfloor$. Despite this intuition, the exact formula to compute the number of blocks is more involved. The details on how the number of blocks can be computed in the presence of weights on the dimensions are out of the scope of this paper and are omitted.

4.3 *Personalization of the degree of anonymity*

In this paper we so far did not consider issues related to the personalization of the degree of anonymity k. Some approaches (e.g., (22)) explicitly allow different users to specify different values of this parameter. A natural question is if the proposed techniques can be applied, and can be considered safe even in this case. Once again, to answer this question it is essential to consider which knowledge an attacker may obtain. The degree of anonymity k desired by each user at the time of a request is not assumed to be known by the attacker in contexts C_{st} and C_{st+g}, hence algorithms that are safe for these contexts remain safe even when the LTS admits different values of k.

However, it may be reasonable to consider contexts in which the attacker may obtain information about k. For example, if the attacker is able to derive information from different requests, he can exploit data mining techniques to derive, with a certain likelihood, the value of k required by each user. This knowledge can then be used to perform new attacks. Since these attacks are based on contexts different from C_{st+g}, the C_{st+g}-safe generalization function presented in this section are not able to provide protection against this new kind of attacks.

A straightforward solution to extend C_{st+g}-safe algorithms to these cases is the following: when a request r needs to be generalized with degree of anonymity k, the anonymity set is computed considering only the users that can possibly issue a request requiring that degree of anonymity. Clearly, the solution is viable only if a limited set of k values is available and a large number of users using each value exists. If this is not the case, more sophisticated strategies need to be devised to obtain C_{st+g}-safe generalization algorithms, and, to our knowledge, this is still an open research issue.

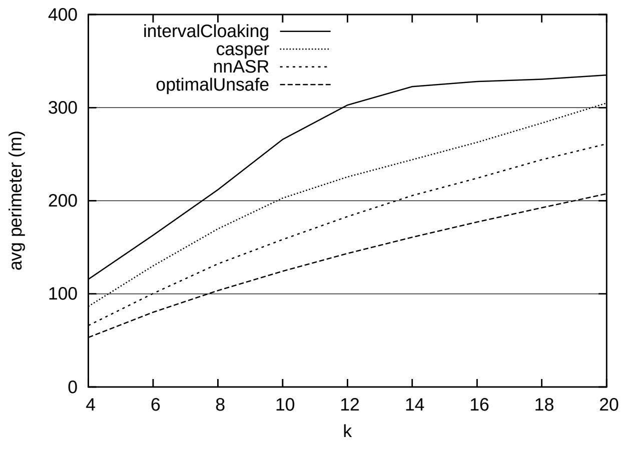
5 Experimental results

We performed an extensive experimental evaluation of the algorithms presented in Section 4 using two synthetic datasets. In both cases, the total area of the considered map is about 100 km^2 and the maximum number of users is $500,000$. In the first dataset, users are uniformly distributed, while in the other, users' locations are generated by the moving object generator developed by Brinkhoff (8) that was set to generate users' locations in the streets of the city of San Francisco. In the experiments with $500,000$ users, the average density of users for km^2 is about $5,000$ which is in the same range of the real density of population in that area. We identified two main parameters for each test: the degree of anonymity k, and the total number p of users.

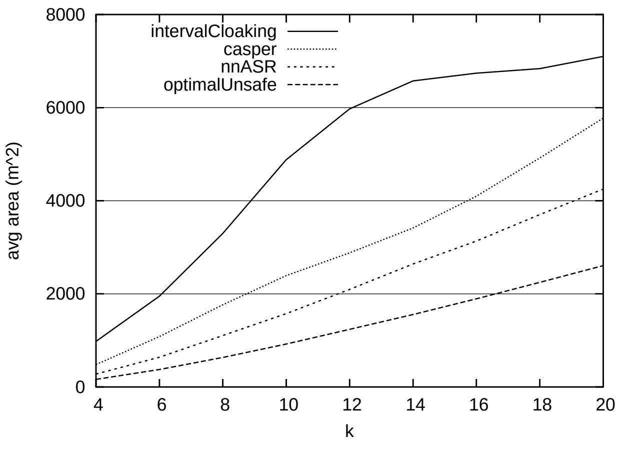
We implemented the algorithms using Java, and we performed our tests on a Linux machine with two 2,4Ghz Pentium Xeon processors and 4GB of shared RAM. All the output values presented in this section are obtained by running $1,000$ tests and taking the average.

In order to compare the regions returned by the generalization algorithms with the smallest possible generalized region, 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 the number of users p; However, we developed several optimization techniques that make the algorithm in most cases computable in time linear 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 C_{st+g}-unsafe algorithms, for quite large values of p and practically relevant values of k. The algorithm we implemented computes, among all possible anonymity sets, the one having the smallest perimeter of the *MBR*. The choice of computing the optimal perimeter, instead of the optimal area, is driven by the fact that if an optimal area is computed then the algorithm often returns a set of users having the same x value (or y value) but that are possibly far from each other.

Figures 6(a) and 6(b) show the average perimeter and area, respectively, of the region returned by four C_{st+g}-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 this is not the case. On average, *nnASR* returns regions having a perimeter 25% larger than the one of the region returned by

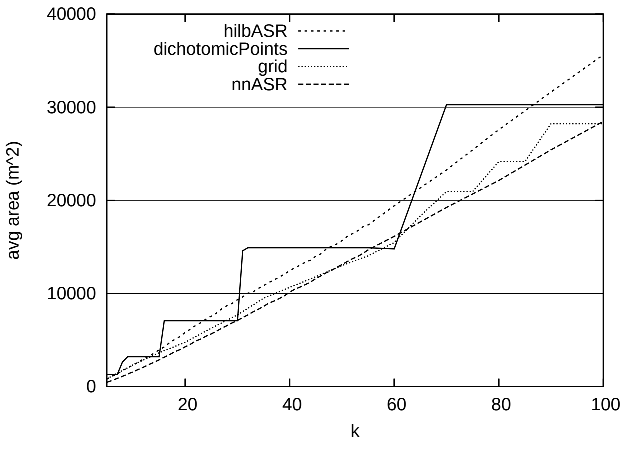


(a) perimeter

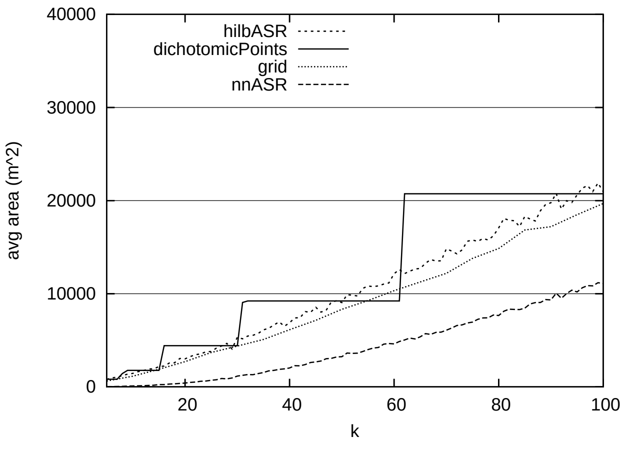


(b) area

Figure 6. Average size of the generalized region with $p = 500,000$ for the C_{st+g}-unsafe algorithms in the non-uniform distribution.



(a) uniform distribution



(b) non-uniform distribution

Figure 7. Average area with $p = 500,000$ for the C_{st+g}-safe algorithms.

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 $p = 500,000$, *nnASR* returns the region with the minimal perimeter in about 26% of the cases, while the percentage drops below 2% for $k = 20$ and the same number of users.

Figure 7(a) and 7(b) consider the C_{st+g}-safe algorithms *hilbASR*, *dichotomicPoints*, *grid*, and one C_{st+g}-unsafe algorithm, *nnASR*, to compare with. They show the average area of the regions returned by these algorithms for different values of k, in both uniform and non-uniform cases. First, we can notice that, in the non-uniform distribution, the algorithms perform better. This is due to the fact that users are located in the streets only and therefore they tend to be closer to each other. We can also observe that in both cases the *grid* algorithm performs significantly better than the other two C_{st+g}-safe algorithms. Comparing *grid* and *nnASR* we can notice that, in the uniform distribution, they perform similarly for values of k up to 60, while for higher values of this parameter, *nnASR* performs slightly better. In the non-uniform case, *nnASR* performs significantly better for every value of k. Indeed, in the non-uniform case it is possible that projecting the positions of a given set of users on one of the axis, some successive values on the axis happen to be far apart from each other. Due to its partitioning strategy, *grid* may have to include the corresponding positions in the same block, while *nnASR*, for the same set of users may be able to include only positions with values on that axis close to each other. This clearly does not happen in the uniform case, since the values in the projection will always have a similar distance. Finally, we can observe that the the growth of the size of the average area obtained using *hilbASR*, *grid* and *nnASR* is almost linear in

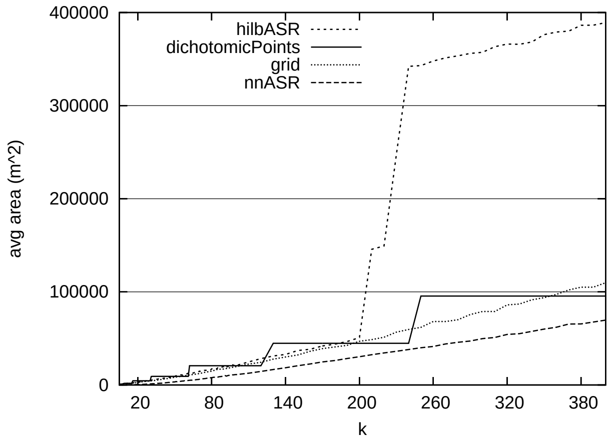


Figure 8. Average area with $p = 500,000$ for the non-uniform distribution and for large values of k.

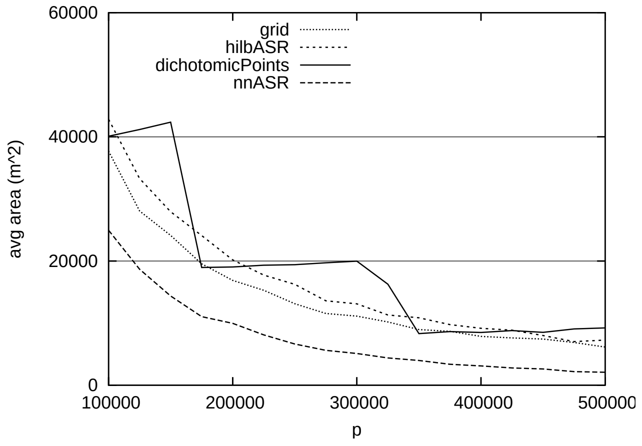


Figure 9. Average area with $k = 40$ for the non-uniform distribution.

k, while the growth obtained using *dichotomicPoints* is constant for certain intervals of k, with changes of values for some values of k. This is due to the fact that *dichotomicPoints* 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.

In order to better illustrate this behaviour, we show in Figure 8 the average area of the *MBR* for high values of k (up to 400). It can be noticed that *hilbASR* does not scale well for high values of k. Though this values of k may be assumed too high 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 (6).

In Figure 9 we fix the value of k to 40 and we show the average area of the generalized region computed by the four algorithms with respect to the considered number of users. As expected, the average area decreases for a growing value of p. A slightly different behaviour can be noticed for the *dichotomicPoints* algorithm: since, as we observed before, the number of divisions performed by *dichotomicPoints* is equal to $\lceil \log(\frac{p}{k}) \rceil$, if we increase the value of p, keeping the same number of divisions, we will have larger anonymity sets, hence larger areas of their *MBR*.

In Figure 10 we show the variance of the area of the *MBR* of the generalized regions computed by the *dichotomicPoints*, *hilbASR*, *grid* and *nnASR* algorithms. The growth of the *hilbASR* variance is not regular. We have observed that, in some cases, the execution of the *hilbASR* algorithm in the non-uniform distribution can result in an area that is up to 30 times larger than the average one. On the other hand, in most of the cases the algorithm generates areas smaller than the average one. The *dichotomicPoints*

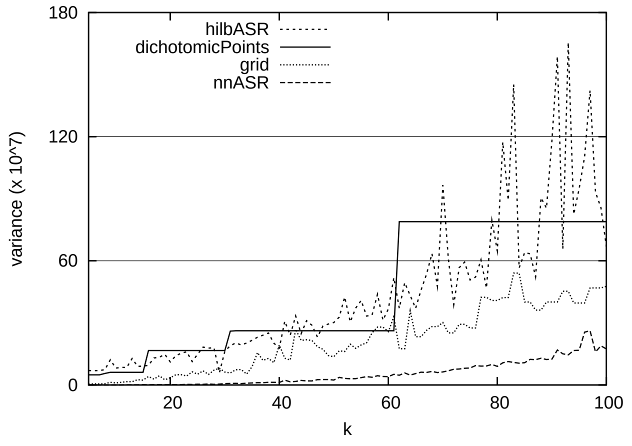
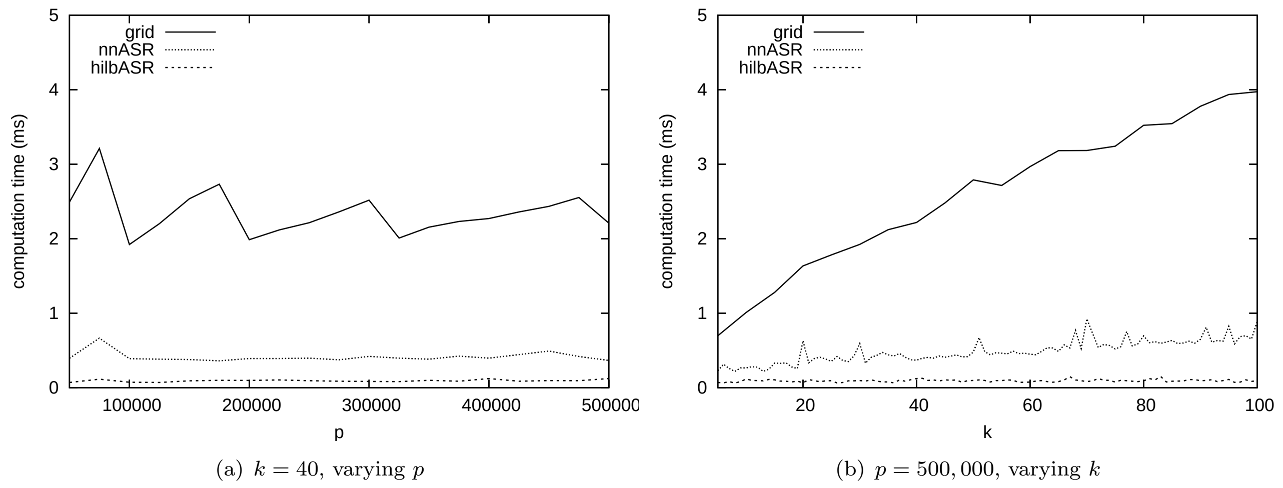


Figure 10. Variance of the algorithms with $p = 500,000$ for the non-uniform distribution.



(a) $k = 40$, varying p

(b) $p = 500,000$, varying k

Figure 11. Average computation time for the non-uniform distribution.

algorithm has a regular but still high variance, while *grid* algorithm has a smaller variance. As expected, the *nnASR* algorithm has a much more regular and a definitely better variance.

Figures 11(a) and 11(b) show the average computation time of the algorithms *nnASR*, *hilbASR* and *grid*. In the former, the value of k is fixed to 40 and the value of p varies from a minimum of 100,000 to a maximum of 500,000. In the latter, p is fixed to 500,000 and k varies from 5 to 100. The results for the *dichotomicPoints* are not reported here, since this algorithm has a computation time about 50 times higher with respect to the other algorithms reported in the two figures. Indeed, for $k = 40$ and $p = 500,000$ the computation time of *dichotomicPoints* is about half a second. We can notice that the computation time of the *nnASR* and *hilbASR* algorithm is less than one millisecond and we did not measure significant variation in this value for different values of k of p. As expected from the complexity analysis, the computation time of *grid* algorithm is affected both by the number of users and the k degree. As described in Section 4, in order to improve the scalability of the algorithm, we partitioned the area in blocks having the same area, and apply the generalization only considering users in the block from which the request was issued. Since in our experimental scenario we considered an area with very high density, we divided the total area in a grid having $\lfloor\sqrt{\frac{p}{20,000}}\rfloor$ rows and colums, which results in a minimum of 20,000 users for each block. It is not difficult to adapt this preprocessing technique to different scenarios using blocks of different sizes based on the population density and other statistical parameters. In Figure 11(a) we can note that, using this technique, the computation time of the *grid* algorithm is not significantly affected by the number of users while it grows linearly with the value of k.

6 Related work

6.1 *Anonymity in databases*

Guaranteeing users' anonymity is a well-known problem for the release of data in database tables (23). In this case, the problem is to protect the association between the identity of an individual and a tuple containing her sensitive data; the attributes whose values could possibly be used to restrict the candidate identities for a given tuple are called *quasi-identifiers* (7; 11).

The idea first proposed in (23) to formally measure the intuitive notion of anonymity of a table relies on the concept of *k-anonymity*. Intuitively, a table is k-anonymous if each tuple is not distinguishable, considering the values in the quasi-identifiers attributes, from at least other $k-1$ tuples.

This research area has been very active in the last years. Two main problems were addressed. First, the specification of efficient algorithms that render an input table into a k-anonymous table and that avoid to generalize or suppress too many values from the input table (1; 2; 13; 18; 21; 23; 24; 25) A second research direction aims to extend the notion of k-anonymity in order to provide privacy protection under different assumptions with respect to the ones considered in (23) (among others, (7; 9; 19; 26; 27; 28)). In particular, Xiao et al. (28), consider the case in which the database table can be *re-published* several times, each time after the insertion or deletion of some tuples. That paper is particularly interesting with respect to this paper because the problem of the re-publication of data is intrinsic in the research area of privacy in LBSs. However, the techniques proposed in that paper cannot be easily applied in the field of LBSs. The main difference, is that in (28) each user can be identified, in different publications, by a single combination of values of quasi-identifier attributes. In practice, for a given user only sensitive data can change in different publications. On the contrary, in LBSs the location of the user can change at each request and therefore the values of the quasi identifying attributes can change each time a request is issued.

6.2 *Spatio temporal anonymity*

To the best of our knowledge, the problem of privacy in LBSs was first presented by Gruteser et al. (15). In that paper, the *IntervalCloaking* algorithm, (see Section 4.1) is proposed as a spatio-temporal generalization technique to guarantee the anonymity of a LBS request.

A different algorithm, called *CliqueCloak* was proposed by Gedik et al. (14). The main difference with respect to the *IntervalCloaking* algorithm is that *CliqueCloak* computes the generalization among the users that actually issue a request and not among the users that are potential issuers. Indeed, *CliqueCloak* collects original requests without forwarding them to the SP until it is not possible to find a spatio-temporal generalization that includes at least k pending requests. Then the requests are generalized and forwarded to the SP. The advantage of the proposed technique is that it allows the users to personalize the degree of anonymity. However, the algorithm has high computational costs and it can be efficiently executed only for small values of k.

Bettini et al. first addressed the problem of anonymity in the case in which the attacker has the ability to recognize traces of requests issued by the same user (6). In that paper the authors show that anonymity is not guaranteed if each request in a trace is generalized using a generalization algorithm for the single-request case. Then, the notion of *Historical k-anonymity* is proposed to define when a trace of requests is anonymous.

In (22) Mokbel et al. propose the *Casper* generalization algorithm for the static case that makes it possible to achieve the personalization of the degree of anonymity. Indeed the algorithm, described in Section 4.1, allows the user to specify a privacy profile composed by two parameters: k and A_{min}. The parameter k is the degree of anonymity while A_{min} is the minimum size of the generalized region that should be forwarded to the SP. In the paper it is not clear how a large value of the parameter A_{min} can enhance better privacy. Indeed, as we formalized in this paper, the probability that an attacker can re-identify the issuer of a request is not affected by the size of the generalized region. A different interpretation of the role of A_{min} is that a large generalized region can help to prevent the release of private information. However, in (22) this intuition is not supported by any formal result.

Beresford in (4) showed a counterexample in which, even if a request is generalized with the *Interval-*

Cloaking algorithm, an attacker can uniquely identify the issuer. The problem, shown in Example 3.15, was called the "outlier problem". The first algorithm that does not suffer the "outlier problem" was proposed by Kalnis et al (16) and was called *hilbASR* (see Section 4). Though the algorithm solves the "outlier problem", no proof of its correctness was provided in (16). This problem was first addressed in (5) and (20) where a preliminary version of the framework presented in this paper was proposed.

Other two alternative results were independently proposed to tackle the "outlier problem". Ghinita et al. (17) proved the correctness of the *hilbASR* algorithm while Chow et al. (10) defined the "k-sharing region" property; Intuitively, a generalization algorithm that has this property does not suffer from the "outlier problem".

The main difference between our approach and the approaches proposed in (17) and (10) is that we formally identify the external knowledge that allows that attack to be performed. Moreover, while the solutions proposed in the other two papers are specific for a given external knowledge, the formalism we propose is flexible enough to apply to different contexts, including the one in which the attacker knows the generalization function.

7 Conclusion and Future Work

In this paper we presented a formal framework to model the problem of users' privacy in LBS. One of the main ideas of the framework is that the context in which privacy protection is to be provided should be explicitly stated. What we call *context* includes what is in other work called *adversary model* or *adversary background knowledge*. We showed that the proposed framework is sufficiently expressive to formalize most of the techniques considered so far in the literature. Indeed, we used the framework to classify existing defense techniques according to the context in which they can be proved to protect users' anonymity.

We also proposed a new algorithm called *grid* and we proved that it protects users' privacy in the most conservative context among the ones considered in the literature. We evaluated existing defense algorithms as well as the algorithm we proposed with extensive experimental study. Through these empirical evaluations, we showed that the *grid* algorithm is strictly better, according to the average size of the generalized area, than the previously available solutions while still being efficiently computable even for large scale applications.

As future work, we intend to further investigate a theoretical measure of the degree of unsafety of generalization algorithms, since we believe that what has been proposed until now is not satisfactory for all applications. We are also already working on formalizing attacks in which the attacker has the ability to recognize traces of requests issued by the same user. These type of attacks clearly require new defense techniques.

Acknowledgments

We would like to thank the reviewers for their helpful comments. This work was partially supported by the NSF grants IIS-0430402, IIS-0430165, CNS-0716567, and CNS-0716575, and by Italian MiUR grant II04C0EC1D.

References

[1] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering. In *Proc. of the 25th ACM symposium on Principles of database systems*, pages 153–162. ACM Press, 2006.

[2] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of the 21st International Conference on Data Engineering*, pages 217–228. IEEE Computer Society, 2005.

[3] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, January–March 2003.

[4] A. R. Beresford. *Location privacy in ubiquitous computing.* PhD thesis, University of Cambridge, 2005.

[5] C. Bettini, S. Mascetti, X. S. Wang, and S. Jajodia. Anonymity in location-based services: towards a general framework. In *Proc. of the 8th International Conference on Mobile Data Management (MDM).* IEEE Computer Society, 2007.

[6] C. Bettini, X. S. Wang, and S. Jajodia. Protecting privacy against location-based personal identification. In *Proc. of the 2nd workshop on Secure Data Management (SDM)*, volume 3674 of *LNCS*, pages 185–199. Springer, 2005.

[7] C. Bettini, X. S. Wang, and S. Jajodia. The role of quasi-identifiers in k-anonymity revisited. Technical Report RT-11-06, DICo, University of Milan, 2006.

[8] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[9] J.-W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *Proc. of Third VLDB Workshop on Secure Data Management*, Lecture Notes in Computer Science. Springer, 2006.

[10] C. Y. Chow and M. F. Mokbel. Enabling private continuous queries for revealed user locations. In *Proc. of Advances in Spatial and Temporal Databases, 10th International Symposium* SSTD 2007, Springer, 2007,

[11] T. Dalenius. Finding a needle in a haystack - or identifying anonymous census record. *Journal of Official Statistics*, 2(3):329–336, 1986.

[12] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *Proc. of the International Conference on Pervasive Services (ICPS)*, pages 88–97. IEEE Computer Society, 2005.

[13] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc of the 22nd International Conference on Data Engineering (ICDE)*, 2006.

[14] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proc. of the 25th International Conference on Distributed Computing Systems (ICDCS)*, pages 620–629. IEEE Computer Society, 2005.

[15] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of the 1st International Conference on Mobile Systems, Applications and Services (MobiSys).* The USENIX Association, 2003.

[16] P. Kalnis, G. Ghinta, K. Mouratidis, and D. Papadias. Preserving location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1719–1733, 2007.

[17] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Prive: anonymous location-based queries in distributed mobile systems. In *Proc. of the 16th international conference on World Wide Web*, pages 371–380. ACM Press, 2007.

[18] A. Machanavajjhala and J. Gehrke. On the efficiency of checking perfect privacy. In *Proc. of the 25th ACM symposium on Principles of database systems*, pages 163–172. ACM Press, 2006.

[19] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *Proc. of the 22nd International Conference on Data Engineering*, 0:24, 2006.

[20] S. Mascetti and C. Bettini. A comparison of spatial generalization algorithms for lbs privacy preservation. In *Proc. of the 1st International Workshop on Privacy-Aware Location-based Mobile Services (PALMS).* IEEE Computer Society, 2007.

[21] S. Mascetti, C. Bettini, X. S. Wang, and S. Jajodia. k-anonymity in databases with timestamped data. In *Proc. of 13th International Symposium on Temporal Representation and Reasoning.* IEEE Computer Society, 2006.

[22] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *Proc. of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 763–774. VLDB Endowment, 2006.

[23] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[24] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[25] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[26] R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang. k-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *Proc. of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 754–759. ACM Press, 2006.

[27] X. Xiao and Y. Tao. Personalized privacy preservation. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 229–240. ACM Press, 2006.

[28] X. Xiao and Y. Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *Proc. of the 2007 ACM SIGMOD international conference on Management of data*, pages 689–700. ACM Press, 2007.

Appendix A: Proofs

A.1 *Proof of Theorem 3.10*

Proof By Definition 3.7, a generalization function $g()$ is a defense function against $UAtt_{C_{st}}$ with threshold $1/k$ if for each original request r, $g(r)$ is safe against $UAtt_{C_{st}}$ with threshold $1/k$. By Definition 3.6, $g(r)$ is safe against $UAtt_{C_{st}}$ with threshold $1/k$ if $UAtt_{C_{st}}(g(r), issuer(r)) \leq 1/k$. Therefore, we only need to show that, for each original request r, $UAtt_{C_{st}}(g(r), issuer(r)) \leq 1/k$.

Since $UAtt_{C_{st}}$ is a uniform attack, by Proposition 3.5 it follows that, for each original request r, $UAtt_{C_{st}}(g(r), issuer(r)) = (1/|AS_{C_{st}}(g(r))|)$. By hypothesis, $g()$ is such that, for each original request r, $g(r)$ is k-anonymous in context C_{st}. By Definition 3.9 this implies that $|AS_{C_{st}}(g(r))| \geq k$, and therefore $UAtt_{C_{st}}(g(r), issuer(r)) \leq 1/k$. □

A.2 *Proof of Theorem 3.13*

Proof To prove the theorem, we only need to show that, for each generalized request r', if $i \in AS_{C_{st+g}}(r')$ then $i \in AS_{C_{st}}(r')$. For each generalized request r', by Definition 3.12, if $i \in AS_{C_{st+g}}(r')$ then there exists an original request r such that $loc_i = r.Sdata$, $issuer(r) = i$ and $g(r) = r'$. Since we assume that for each generalization function $r.Sdata \in g(r).Sdata$, then $loc_i \in r'.Sdata$. Finally, since $loc_i \in r'.Sdata$, by Definition 3.8, $i \in AS_{C_{st}}(r')$. □

A.3 *Proof of Theorem 3.16*

Proof By Definition 3.7, Gen_{st+g} computes a defense function against $UAtt_{C_{st+g}}$ with threshold $1/k$ if, for each original request $r \in R$, $r' = Gen_{st+g}(r, k)$ is a safe request against $UAtt_{C_{st+g}}$ with threshold $1/k$. By Definition 3.6, r' is a safe request against $UAtt_{C_{st+g}}$ with threshold $1/k$ if $UAtt_{C_{st+g}}(r', issuer(r')) \leq 1/k$. By Proposition 3.5 and Definition 3.8, $UAtt_{C_{st+g}}(r', issuer(r')) \leq 1/k$ if

$$|AS_{C_{st+g}}(r')| = |\{i \in I | \exists \bar{r} \in R \text{ s.t. } loc_i = r.Sdata, issuer(\bar{r}) = i \text{ and } g(\bar{r}) = r'\}| \geq k$$

We prove that this inequality holds by proving that (1) the set AS returned by Algorithm *2* has cardinality at least k and (2) for each user i in AS, there exists a request $\bar{r}$ in R issued by i such that $Gen_{st+g}(\bar{r}, k) = r'$.

(1) Since at Line 1 of Algorithm *2* AS is set to I that has cardinality at least k and AS is reassigned to a block only if each block has cardinality at least k, the cardinality of the set AS returned by the algorithm is at least k.

(2) Let i be a user in the resulting AS and $\bar{r}$ a potential original request issued by i. Formally, $\bar{r}$ is such that $issuer(\bar{r}) = i$, $\bar{r}.Sdata = loc_i$, and $\bar{r}.SSdata = r.SSdata$. Moreover, let $AS(j)$ and $\overline{AS}(j)$ denote the

values of the variables AS in the j-th iteration of $Gen_{st+g}(r,k)$ and $Gen_{st+g}(\bar{r},k)$, respectively. We show by induction on the number of iterations that $AS(j)=\overline{AS}(j)$ for each iteration j.

Induction basis: In the first iteration, $AS(1)=\overline{AS}(1)=I$.

Induction: Assume at iteration j, $AS(j)=\overline{AS}(j)$. We need to show that either both executions terminate at that iteration, or $AS(j+1)=\overline{AS}(j+1)$. Indeed, since $AS(j)=\overline{AS}(j)$, the partition procedure is deterministic and has the value of the variable AS as the only input, it follows that the same partition $\{AS_1,\ldots,AS_n\}$ is computed for $AS(j)$ and for $\overline{AS}(j)$. Consequently, if there exists a block with cardinality less than k, both executions terminate, returning the same value for AS. On the contrary, if all the blocks have cardinality larger than or equal to k, then $AS(j+1)$ is the block in $\{AS_1\ldots,AS_n\}$ that contains $issuer(r)$ and $\overline{AS}(j+1)$ is the block in the same set that contains i. We claim that $AS(j+1)$ must contain i. Indeed, we know that the resulting AS of the algorithm must be a subset of $AS(j+1)$ due to successive partitioning method of the algorithm and we assumed that i is in the resulting AS. Now since $\overline{AS}(j+1)$ also contains i, we know $AS(j+1)=\overline{AS}(j+1)$ due to the fact that $\{AS_1\ldots,AS_n\}$ is a partition of $AS(j)=\overline{AS}(j)$. □

A.4 *Proof of Theorem 4.1*

Proof Analogously to the proof of Theorem 3.16, we need to prove that

$$|\{i\in I|\exists\bar{r}\in R \text{ s.t. } loc_i=\bar{r}.Sdata, issuer(\bar{r})=i \text{ and } hilbASR(\bar{r},k)=r'\}|\geq k,$$

where $Gen_{st+g}()=hilbASR()$. A detailed description of *hilbASR* can be found in (16). To prove that this inequality holds, we only need to show that (1) the set AS returned by *hilbASR* has cardinality at least k and (2) for each user i in AS, there exists a request $\bar{r}$ in R issued by i such that $Gen_{st+g}(\bar{r},k)=r'$.

(1) This follows from the values given to *start* and *end* (see (16)). Indeed, it can be easily seen that $|AS|\geq end-start+1$ that is equal to $start+k-1-start+1=k$.

(2) Since AS is the only parameter used in the computation of the partition, the proof is analogous to point (2) in the proof of Theorem 3.16.

□

A.5 *Proof of Theorem 4.3*

Proof Analogously to the proof of Theorem 3.16, we need to prove that

$$|\{i\in I|\exists\bar{r}\in R \text{ s.t. } loc_i=\bar{r}.Sdata, issuer(\bar{r})=i \text{ and } dichotomicPoints(\bar{r},k)=r'\}|\geq k,$$

To prove that this inequality holds, we only need to show that (1) the set AS returned by Algorithm *3* has cardinality at least k and (2) for each user i in AS, there exists a request $\bar{r}$ in R issued by i such that $Gen_{st+g}(\bar{r},k)=r'$.

(1) At each iteration of the main loop, if $|AS|\geq 2\cdot k$, AS is partitioned in two blocks, otherwise the algorithm terminates. The two blocks in which AS is partitioned have cardinality $\lfloor|AS|/2\rfloor$ and $\lceil|AS|/2\rceil$, respectively. Since AS is partitioned only if $|AS|\geq 2\cdot k$, the two blocks have at least cardinality k. Therefore, the final AS set has at least cardinality k.

(2) Since AS is the only parameter used in the computation of the partition, the proof is analogous to point (2) in the proof of Theorem 3.16. □

A.6 *Proof of Theorem 4.5*

Proof Analogously to the proof of Theorem 3.16, we need to prove that

$$|\{i\in I|\exists\bar{r}\in R \text{ s.t. } loc_i=\bar{r}.Sdata, issuer(\bar{r})=i \text{ and } Grid(\bar{r},k)=r'\}|\geq k,$$

To prove that this inequality holds, we only need to show that (1) the set AS returned by Algorithm *4* has cardinality at least k and (2) for each user i in AS, there exists a request $\bar{r}$ in R issued by i such that $Gen_{st+g}(\bar{r}, k) = r'$.

(1) In each of the two iterations of the Algorithm *4*, variable AS is reassigned to a set with cardinality at least equal to the current value of the variable upb. Indeed, the cardinality of AS is given by $end - start + 1$; When the condition of the **if** statement of line 13 is verified, then $end - start + 1 = upb$. Otherwise,

$$end - start + 1 = |AS| - 1 - (nob - 1)upb + 1 = |AS| - nob \cdot upb + upb \geq |AS| - |AS| + upb = upb$$

During the first iteration $upb = \lfloor |I|/nob \rfloor$ where $nob = \left\lfloor \sqrt{|I|/k} \right\rfloor$ (note that the value of nob is fixed before the main cycle and is never changed). During the second iteration, upb is set to $\lfloor |AS|/nob \rfloor$. Since $|AS|$ is at least the value that upb has during the first iteration, then $upb \geq \left\lfloor \frac{\lfloor |I|/nob \rfloor}{nob} \right\rfloor$ and hence, since $k \geq 1/h$, we need to prove that $\left\lfloor \frac{\lfloor |I|/nob \rfloor}{nob} \right\rfloor \geq k$ This inequality is equivalent to $\frac{\lfloor |I|/nob \rfloor}{nob} \geq k$ and hence to $\lfloor |I|/nob \rfloor \geq k \cdot nob$.

This inequality follows from the fact that, by definition, $nob = \left\lfloor \sqrt{|I|/k} \right\rfloor$, hence $nob \leq \sqrt{|I|/k}$ and therefore, since $nob > 0$, $k \cdot nob^2 \leq |I|$. Consequently, $|I|/nob \geq k \cdot nob$ and finally $\lfloor |I|/nob \rfloor \geq \lfloor k \cdot nob \rfloor = k \cdot nob$.

(2) Since AS is the only parameter used in the computation of the partition, the proof is analogous to point (2) in the proof of Theorem 3.16. □