TRANSACTIONS ON DATA PRIVACY 7 (2014) 131–163

SafeBox: adaptable spatio-temporal generalization for location privacy protection

Sergio Mascetti, Letizia Bertolaja, Claudio Bettini

Università degli Studi di Milano, Computer Science Dep., EveryWare Lab.

E-mail: {sergio.mascetti, letizia.bertolaja, claudio.bettini}@unimi.it

Abstract. Spatial and temporal generalization emerged in the literature as a common approach to preserve location privacy. However, existing solutions have two main shortcomings. First, spatio-temporal generalization can be used with different objectives: for example, to guarantee anonymity or to decrease the sensitivity of the location information. Hence, the strategy used to compute the generalization can follow different semantics often depending on the privacy threat, while most of the existing solutions are specifically designed for a single semantics. Second, existing techniques prevent the so-called *inversion* attack by adopting a top-down strategy that needs to acquire a large amount of information. This may not be feasible when this information is dynamic (e.g., position or properties of objects) and needs to be acquired from external services (e.g., Google Maps).

In this contribution we present a formal model of the problem that is compatible with most of the semantics proposed so far in the literature, and that supports new semantics as well. Our *BottomUp* algorithm for spatio-temporal generalization is compatible with the use of online services, it supports generalizations based on arbitrary semantics, and it is safe with respect to the inversion attack. By considering two datasets and two examples of semantics, we experimentally compare *BottomUp* with a more classical top-down algorithm, showing that *BottomUp* is efficient and guarantees better performance in terms of the average size (space and time) of the generalized regions.

1 Introduction

Emerging applications in the area of mobile and pervasive computing have significantly increased the risk of privacy threats by the uncontrolled release of information about the whereabouts of individuals. This information is not only released by the individuals themselves while using their mobile phones or wearable sensing technology, but in some cases it is published by other users in social applications or transferred by providers to third parties. The collection of information about the presence of individuals in certain places at given times can lead to information about their movements, behavioral habits, and can potentially be used for unsolicited advertisement, discrimination and even stalking attacks. This risk is not only theoretical: for example, the work of Fattori et al. shows how it is possible (and relatively easy) to violate users' privacy in existing real-world friend-finder services [7].

An extensive literature exists about location privacy and protection techniques [5, 17]. Spatial and temporal generalization is a common approach that has also been implemented in prototypes and applications. Spatial generalization decreases the precision of location information, hence introducing uncertainty about the actual position of the user within

the reported geographical area. Temporal generalization introduces uncertainty about the precise time of presence of the user in the reported area, and is usually implemented by delaying the release of the spatio-temporal information and providing a temporal interval or using a coarser granularity instead of a precise timestamp. In this paper we call *SafeBox* the spatio-temporal region used to generalize a source point representing the user's exact location and time of presence.

The Problem

A crucial issue for this approach is deciding how large the SafeBox should be in order to avoid a privacy breach. This decision is not only important for privacy protection, but it also has an impact on the performance and precision of the service being used. The minimum size of the SafeBox is actually dependent on the context, including the individual's privacy preference, the application being considered, the time and place of service requests, the adversary model and other parameters. For example, solutions aimed at protecting identity privacy in LBS and adopting techniques inspired by *k-anonymity* (e.g., [8, 12, 11]), have the goal to identify *the smallest* SafeBoxes that contain at least k other users in addition to the issuer of a LBS request that may potentially use the same service; by releasing the SafeBox instead of the exact position, a form of anonymity is enforced, since the released information by itself cannot be used to re-identify the individual, even in the case in which the adversary knows the identity of all the users in the reported area. Other generalization solutions, not focused on identity privacy but more on hiding the presence of an individual in a potentially sensitive place (e.g., [6]) have a different optimization criteria for the dimension of their SafeBoxes. In some cases they want *the smallest* regions that include at least k other venues, as pubs, shops, offices, in order to enforce uncertainty about the actual venue where the user is/was located. Increasing the temporal size helps keeping the spatial size small but it should satisfy the real-time constraints that the considered application may have.

Each of the proposed techniques is somehow specialized to optimize the generalization with respect to the specific semantics of privacy preservation (counting users, venues, categories of venues, ...). Hence, one problem we would like to address in this paper is to have a generalization scheme solution that is parametric with respect to a *counting function*, and hence independent from the actual semantics of privacy preservation.

A second problem with current solutions is related to the method to actually compute the SafeBox. In order to avoid the so called "inversion" or "reciprocity" attack [12, 11], most of the proposed techniques for spatial generalization, compute the SafeBox by partitioning the whole space and explore it with a *top-down* strategy. The whole spatial domain is recursively partitioned evaluating at each step if the area containing the user's location satisfies the counting function, and returning that area if further partitioning violates the constraints. It is somehow assumed that there is no cost in the access to the location information about the objects to be counted. It is also assumed that counting for large regions is feasible and does not incur in significant overhead. This assumption may be reasonable if the generalization is done, for example, by a service provider that has access to continuously updated user location information and the objects to be counted are indeed users, but in general, computing the counting function may result in a costly operation to an external server that has to be frequently repeated when the objects to be counted are not static, or change their properties in time. Indeed, a natural solution is to use online services that, given an area and a category, return objects of that category with their location and properties. For example, the Google Places API provides two methods for searching for places on a map given a source point. However, each call can return at most 200 results

and separate calls are needed to get details (e.g., the opening hours). Each call can require up to 2 seconds to have a response, depending on the connection, and, moreover, only 1000 calls can be performed daily without special permissions. In this scenario, evaluating privacy conditions based on counting may become impractical with generalization strategies that operate top-down, because counting for large areas may be very time-consuming, if possible at all.

Hence, the second problem we are addressing in this paper is devising a new method to compute the SafeBox, compatible with the typical constraints involved in querying external services to obtain updated information on geo-referenced objects.

Contribution

Our solution to the second problem described above is a generalization algorithm that operates *bottom-up*: It builds the SafeBox starting from the actual position and timestamp of the user, and, by adopting a specific data structure, recursively enlarges the spatio-temporal region until the counting constraints are satisfied, while maintaining protection against the "inversion" attack.

The main contributions of this paper are the following:

- By providing a general notion of object counting function supporting different semantics we capture in a single problem formalization several location privacy problems previously considered in the literature and enable capturing privacy preferences not considered in previous generalization approaches.
- We design a generalization algorithm supporting arbitrary counting functions that operates bottom-up, enabling the verification of privacy conditions through public online services. To our knowledge this is the first safe generalization algorithm adopting a bottom-up strategy.
- We implemented the *BottomUp* algorithm and applied it to two different datasets, presenting a detailed comparison in terms of precision and performance, both with its top-down counterpart and with algorithms proposed in related work. The results confirm that our algorithm is effective, superior, and possibly the only alternative when access to updated external data is limited.

The rest of the paper is structured as follows. In Section 2 we discuss related work. In Section 3 we formalize the privacy problem and we define the two semantics of the counting function that we use in our experimental evaluation. Section 4 presents the new *BottomUp* algorithm and compares it with one following the more traditional top-down approach. The experimental evaluation is reported in Section 5, and Section 6 concludes the paper.

2 Related Work

Some contributions in the literature adopted spatio-temporal generalization to enforce users' anonymity while others used this technique to decrease the sensitivity of location information. One of the first techniques aimed at guaranteeing anonymity through spatial and temporal cloaking was proposed by Gruteser and Grunwald [10]. The idea is to guarantee a form of k-anonymity in such a way that the issuer of a location-based service request cannot be identified.

One problem of the solution proposed by Gruteser and Grunwald is that the technique is unsafe if the generalization function is known to the adversary. This problem, called

"inversion" or "reciprocity", has been addressed in [12, 11]. These two papers propose two analogous formal properties of the generalization function. Intuitively, a generalization function $\mathcal{G}$ meets these properties if each point contained in any generalized region r computed by $\mathcal{G}$ is generalized to r itself. The two papers prove that, if a generalization algorithm meets this property, then it is safe with respect to the inversion problem. In this contribution we propose a different property (see Section 3) that intuitively states that a generalized region r computed by a generalization function $\mathcal{G}$ should contain at least k objects that, if used as source points, are generalized to r itself. The difference is that, in this new property, we only require that some points of r (i.e., not necessarily all of them) generalize to r itself. Hence this is clearly a looser property to meet but, as we prove in this contribution, it still guarantees the safety of the generalization algorithms with respect to the inversion problem. This new definition is required by our *BottomUp* algorithm that does not meet the properties defined in [12, 11] but instead meets the new property defined in this contribution and hence it is safe with respect to the inversion problem. The main problem with the generalization algorithms proposed in [12, 11] is that both require the knowledge of all the objects to be counted and hence are impractical when retrieving this information is costly, like, for example, in one of our experimental settings (see Section 5.3.4).

Other solutions proposed in the literature present the problem above. Among the others we can mention the technique proposed by Gedik et al. that has the advantage of allowing each user to choose a personalized value of k [8], the solution by Abul et al. that makes it possible to anonymize a dataset of trajectories [1] and the solution by Mascetti et al. that addresses the issue of anonymity in location based services when different requests can be associated to the same user [13].

In the contribution by Ghinita et al. the objective of the spatial generalization is not to provide anonymity rather to decrease the sensitivity of the location information [9]. The technical problem is that, given two generalized spatio-temporal regions representing the location of a user, an adversary can be able to exclude part of them as possible user position if the maximum velocity for that user is known. The solution by Ghinita et al. does not investigate how the generalized spatio-temporal regions should be created, which instead is the focus of our contribution. We believe that extending our solution with a technique like the one presented in [9] is an interesting future work.

Other contributions share the same objective of decreasing the sensitivity of the location information. Some of them are specifically designed for the so-called friend-finder services [14, 16, 18] while others focus on how to let the user specify the desired level of privacy protection [2]. None of these contribution focus on how to create generalized spatio-temporal regions that contains a minimum number of objects.

Some of the solutions proposed in the literature are aimed, at the same time, at guaranteeing anonymity and at decreasing the sensitivity of the location information [15, 3, 4]. In particular Bamba et al. suggest two generalization techniques (called "Top-Down grid cloaking" and "Bottom-Up grid cloaking") that enforce k-anonymity, l-diversity and a minimum size of the generalized area [3]. Unfortunately both techniques suffer from the inversion problem and hence are safe only if the generalization technique is not known to the adversary. Our previous work has the same problem [4]. Indeed in [4] we define three spatial granularities to support privacy preservation. In particular the $\mathcal{I}ncognitus$ granularity is constructed with a bottom-up approach in such a way that each granule contains at least k objects. The solution can be applied to users (hence providing k-anonymity) and to points of interests (hence guaranteeing a decrease in the sensitivity of the location information). Unfortunately, as we detail in Section 4, also this technique is subject to the inversion

problem.

Finally, the solution by Damiani et al., takes into account the "semantic location" i.e., specific locations where the user does not want to be reported [6]. This is an innovative approach and it has two main differences with respect to ours. First, the generalization applies to the entire map and is computed off-line. Vice versa, in our solution we generalize the location of the user on-the-fly, so the generalization function can use dynamic information. The second difference is that in the solution by Damiani et al. a user's location is generalized only if it falls into an "obfuscated location" i.e., a sensitive location or its surroundings. This can lead to disclose which are the sensitive locations of a user, which we believe should be considered private information. To avoid this problem, in our solution we propose generalization functions that can be used to generalize requests from every location.

3 Problem Formalization

We address the problem of *generalizing* the information about a specific location and timestamp into a geographical area and a time interval so that the resulting spatio-temporal information is still useful to obtain geo-referenced and timely services but not sensible anymore in terms of privacy. Since privacy is a subjective matter, the way generalization occurs not only has to be safe but should also be adapted to the user preferences. Our framework captures all preferences that can be expressed as a guarantee of presence in the released area of enough elements to sufficiently decrease the sensitivity of the spatio-temporal information being released.

In the following of this section we first formally describe the general problem and then discuss the properties of the SafeBoxes that our techniques can return, depending on the different semantics associated to the function used to count the elements they contain.

3.1 Adversary model

The generalization techniques proposed in this contribution can be used to protect users' privacy in different system architectures. Indeed, the generalization function can be computed either by the user's mobile client or by a trusted generalization server. In both cases, the source point (i.e., the exact user's position at a certain time) is not disclosed to any non trusted entity. Instead, the generalized location is disclosed.

The "adversary" is any entity that can potentially have access to the generalized spatio-temporal location. It can be, for example, a provider of a Location Based Service (LBS), an eavesdropper that intercepts the communication towards the LBS service provider, a hacker that violates the service provider system hence acquiring its stored data or even a govern entity that forces the service provider to disclose its stored information.

Given this general definition, a central aspect to correctly model the problem is to define which knowledge the adversary can use to violate user's privacy (this is sometime referred to as "background knowledge" in the literature). In this paper we assume that the adversary has the knowledge to do the inversion attack that, as we formalize in the following of this section, requires the knowledge of the generalization algorithm, its input parameters (with the exception of the source point) and the counting function. In a real-world generalization service the generalization algorithm would probably be known because the "security-by-obscurity" paradigm has proved to be not effective. For what concerns the input parameters (e.g., the value of k representing the minimum number of objects in each

SafeBox), they can either be system parameters (hence easily discovered by an adversary) or, more likely, user-defined parameters. In the latter case it is still possible for an adversary to infer their value, possibly with some form of approximation (consider Example 1). Finally, for what concerns the knowledge of the counting function, in some case this can be public information (as in Example 1) and, if not, it can be inferred, possibly introducing some approximation in the computation, like in Example 1.

Example 1. Alice uses a privacy-aware LBS. Before issuing any service request, the client generalizes Alice's spatio-temporal location so that it contains at least k open shops where k is a user-defined parameter having values in $[2, 20]$.

Suppose that an adversary can observe a request issued by Alice from a generalized spatio-temporal region A. Since in A there are 6 open shops, the adversary can exclude that the parameter k chosen by Alice is larger than 6. The adversary can also compute that, for any value of k in $[2, 4]$, any request issued from a source point $p \in A$ would be generalized to region smaller than A. Hence the value of k is either 5 or 6.

Now, suppose that the generalization algorithm used to generate A is not safe. It could happen that, for a given point $p \in A$, a request issued from p with value of k equals to 5 returns an area different from A and that the same holds for a request issued from p with $k = 6$. In this case the adversary can exclude p from $I(A)$ even without knowing the exact value of k.

It is important to note that in this paper we consider the spatio-temporal generalization of single requests and we do not address the problems arising when correlation among different requests is possible. Consequently, the direct application of our techniques is subject to two attacks known in the literature.

The first is the "velocity attack" that can lead the adversary to exclude the presence of the user in a given area at a given time, hence possibly restricting the generalized spatio-temporal region [9]. Since at the moment our generalization algorithms do not provide protection with respect to the velocity attacks, they should not be used in case of continuous disclosure of location information. Indeed velocity attack is ineffective if the location information is sporadically disclosed.

The second attack is aimed at violating "historical k-anonymity" and takes place when the counting function is used to count the users and is aimed at guaranteeing the issuer's anonymity [13]. In this case, if it is possible to "link" different generalized regions to the same (anonymous) user, the adversary can intersect the "anonymity sets" hence possibly restricting the possible identity of the user to a set with cardinality smaller than k. Since our generalization algorithms do not take this attack into consideration, if the counting function is used to count users with the aim of providing anonymity, it should be guaranteed that no set of generalized regions can be associated to the same user for example by artificially changing the IP address used in the corresponding service requests.

3.2 Basic definitions

We assume that users and (possibly moving) objects are located in a finite bi-dimensional spatial domain S and we consider their positions in a finite interval of time T. We denote with S_1 and S_2 the two dimensions of S and with p a spatio-temporal point (or "point", when no confusion arises). Formally, $p \in S \times T = S_1 \times S_2 \times T$.

Our goal is *generalizing* any point p (called "source point") into a three-dimensional area with certain properties. Formally, a *generalization function* $\mathcal{G}$ is a partial function that, given a source point p returns a three dimensional area A such that $p \in A$.

One of the required properties for the generalization is to guarantee that the resulting area "contains" at least k objects. We use the function Ω to count the number of objects contained in a given area. Formally, given a set of (possibly moving) objects and a spatio-temporal area A, $\Omega(A)$ is a non-negative integer value representing the number of spatio-temporal points corresponding to positions and associated timestamps of the objects in A. The counting function $\Omega(A)$ can also be applied when A is a set of possibly non-contiguous spatio-temporal points. The specification of Ω includes the set of objects to be considered (e.g., users, shops, taxis, etc.) as well as the actual semantics of the counting operator. In Section 3.3 we report two examples of its semantics. Note that the source point being generalized can be the position of one of the objects (as in the case of the source point is the position of a user and the objects are all users) but can also be an unrelated point (as in the case of objects being pubs and the user not being positioned in any of them). Note also that the results we present in this paper assume only that the counting function is monotonic with respect to the areas.

Definition 2. A counting function Ω is monotonic if, for each pair of spatio-temporal areas A and A' such that $A \subseteq A'$ it holds that $\Omega(A) \leq \Omega(A')$.

Monotonicity captures an intuitive property. For example, if in the main square of a city there are 100 people at given moment, by considering at the same moment a larger area that includes the square, we will count 100 or more people.

When a spatio-temporal area contains at least k objects, we say it is a "SafeBox". Intuitively, the user considers herself to be "safe" by releasing this spatio-temporal information because the source point p can be confused with the positions and timestamps associated to at least k objects.

Definition 3. Given a non-negative integer k and a counting function Ω, a spatio-temporal area A is a *SafeBox* if $\Omega(A) \geq k$.

As observed in the literature [11, 12], even if a generalization function returns a SafeBox according to Definition 3, an adversary may still be able to rule out some of the objects considered in the counting if he knows the generalization function itself, since he may re-apply the generalization function to each candidate source point and compare the result with the area that has been released. In principle, by knowing the generalization function the adversary may also be able to identify the source point p though the so called *inversion attack* [12]. Consider Example 4.

Example 4. Let's consider Figure 1: the spatial and temporal domain is partitioned into four areas. The number in the top right of each area represents the value of $\Omega()$. The objective of the generalization is to have a SafeBox with at least 8 objects. Let's consider a naive generalization function that, given any source point in A_2, generalizes it to A_2. Similar for A_3. Since $\Omega(A_1)$ is less than 8, A_1 is not a SafeBox and hence the algorithm generalizes any source point in A_1 to the SafeBox $A_1 \cup A_2$. Similarly, any source point in A_4 is generalized to $A_3 \cup A_4$.

Now, consider an adversary that knows the generalization function and the values of Ω for each area. If this adversary observes the SafeBox $A_1 \cup A_2$ than he can exclude that the source point is in A_2, because in this case the generalization would be A_2 itself. Consequently the adversary infers that the source point is in A_1 that, however, is not a SafeBox.

In order to contrast this inversion attack we first define the *inversion* function that, intuitively, given a generalized area $\mathcal{G}(p)$, identifies all potential source points.

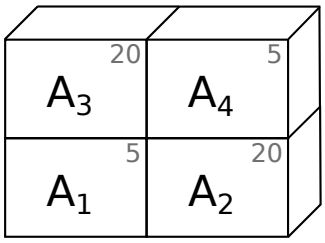


Figure 1: Spatio temporal domain partitioned into four areas.

Definition 5. Given an area A and a generalization function $\mathcal{G}$, the *inversion* function I is defined as:

$$I_{\mathcal{G}}(A) = \{p \in A | \mathcal{G}(p) = A\}$$

When no confusion arises, we simply denote $I(A)$ omitting the generalization function.
We can now define the notion of safety for generalization functions.

Definition 6. Given a non-negative integer k and a counting function Ω, a generalization function $\mathcal{G}$ is *safe* if, for each spatio-temporal point p such that $\mathcal{G}(p)$ is defined, it holds that:

$$\Omega(I(\mathcal{G}(p))) \geq k$$

Example 7. Let's continue with Example 4. We show that the generalization function is not safe according to our model. Indeed, for any point p in A_1, the generalization of that point is $A_1 \cup A_2$. The inversion of $A_1 \cup A_2$ is A_1 (indeed the generalization of any point in A_2 is A_2 itself). Consequently, $\Omega(I(\mathcal{G}(p))) = \Omega(A_1) = 5 < 8$. Consequently, by applying Definition 6, the generalization function is not safe, in accordance with the intuition presented in Example 4.

Let's now consider a different generalization function that generalizes any source point of A_2 in A_2 and similar for A_3. Also, the generalization of any source point in A_1 or A_4 is the entire spatio-temporal domain (i.e., A). This is actually a safe generalization function. Indeed, for any source point p in A_2 it holds that its generalization is A_2 and also that $I(A_2) = A_2$. Consequently $\Omega(I(A_2)) \geq 8$. Similar for A_3. Vice versa, for any point p in A_1 and A_4, the generalization yields A. In this case $I(A) = A_1 \cup A_4$. Hence we have that $\Omega(I(\mathcal{G}(p))) = \Omega(A_1 \cup A_4) = 10 \geq 8$. Note that in this last case (i.e., any source point in A_1 or A_4), it is actually possible to use the inversion attack to restrict the area (from A to $A_1 \cup A_4$), but this does not affect the safety of the technique, since $A_1 \cup A_4$ still contains a sufficiently large number of objects.

Property 1. Any safe generalization function $\mathcal{G}$ returns a SafeBox for any source point p such that $\mathcal{G}(p)$ is defined.

Proofs of formal results are reported in Appendix A.

3.3 Supporting different semantics for the counting function Ω

As specified in Section 3.2, our formal framework only requires the counting function to be monotonic. This weak requirement makes it possible to express most of the semantics proposed in the literature within this model. For example, by counting the users it is possible to enforce a form of k-anonymity while, by counting the shops, it is possible to decrease

the sensitivity of the location information. Also, by counting different types of shops, it is possible to enforce a property similar to l-diversity.

In the following we first specify the *appearance* semantics (that can be used for example to guarantee k-anonymity) and then we specify the *persistence* that is original, to the best of our knowledge.

3.3.1 Appearance semantics

This semantic captures the counting of distinct objects that happen to be within a spatial area A_S in any time instant during a temporal interval A_T. Each object is "counted" if it happens to be located within A_S at least during one time instant of A_T. Each object is counted at most once, independently from how much time it spends within A_S and even if the object enters and exits several times from A_S during A_T.

In order to define this semantic, we first introduce the function $loc()$ that we use to model the position of an object at a given time instant.

Definition 8. Given the set O of objects we define a partial function $loc : O \times T \rightarrow S$ such that, $loc(o, t)$ is the spatial position of object o at time t.

We are now ready to define the *appearance counting semantics*.

Definition 9. Given the set of objects O, an area A and its projections A_S, A_T on the spatial and temporal domain, respectively, the *counting function* Ω with *appearance semantics* is defined as follows:

$$\Omega(A) = |\{o \in O \text{ s.t. } \exists t \in A_T \text{ with } loc(o, t) \in A_S\}|$$

Example 10. Suppose that A_S is a city's park and A_T is from 2 pm until 2.30 pm. Given that O is the set of users of a location based service, the $\Omega()$ counting function with appearance semantics counts how many of these users report their location in the park at least once between 2 pm until 2.30 pm.

Any specification of the $\Omega()$ counting function should be shown to be monotonic for our algorithms to be sound.

Property 2. The counting function $\Omega()$ with appearance semantics is monotonic.

3.3.2 Persistence semantics

According to the appearance semantics each object is counted once independently on how long it has been located within the spatial area A_S during the time interval A_T. In some applications it can be desirable to count more than once those objects that are located in A_S for a "sufficiently long time" during A_T. This semantics captures the intuition that a spatio-temporal area containing some shops for a period of 2 hours provides more privacy protection than a spatio-temporal region that contains the same shops but that has a duration of 10 minutes.

To formalize this semantics we adopt the concept of a *persistence interval* defined as a span of time obtained by partitioning the time domain into intervals with a fixed time duration. For example, if we take 1 hour as the persistence interval duration and we start the persistence intervals at the beginning of a day, the span of time [2014-01-01:08:00, 2014-01-01:09:00) is one of these persistence intervals.

Given a persistence interval duration D, the intuition of the *persistence semantics* is to count, for each object, for how many persistence intervals during A_T that object is located in A_S.

Definition 11. Given the set of objects O, an area A with its projections A_S, A_T on the spatial and temporal domain, respectively, and I the set of persistence intervals with duration D, the *counting function* Ω with *persistence semantics* is defined as follows:

$$\Omega(A) = \sum_{o \in O} |\{i \in I \text{ s. t. } \exists t \in (i \cap A_T) \text{ and } loc(o,t) \in A_S\}|$$

Example 12. Suppose we are computing $\Omega()$ for the center of Milan, for the interval from $7pm$ to $11pm$ of a given day, counting the number of open pubs. A pub that is always open in this time interval is counted 4 times if the persistence interval duration D is 1 hour. If D is 15min it will be counted 16 times, but it will be counted 14 times if D is 15 minutes and it closes at $10{:}30$. Intuitively, the pub may be a possible location for the user in each of the persistence intervals contained in the considered temporal interval if it is actually open at that time. The condition on the opening time is captured in Definition 11 by the predicate $loc(o,t) \in A_S$. The number of persistence intervals intuitively gives a value for a "temporal obfuscation" metrics.

Property 3. The counting function $\Omega()$ with persistence semantic is monotonic.

4 SafeBox computation

In this section we present two safe generalization algorithms that share the main data structure, that we call *generalization tree*, but have a different conceptual approach to the generalization process.

The *TopDown* algorithm starts by considering the entire space and time (the "top") and then "moves down" from the root of the generalization tree searching for a node corresponding to the "smallest" spatio-temporal region that guarantees the safety property of the algorithm. In contrast, the *BottomUp* algorithm starts from the leafs of the generalization tree, which intuitively correspond to small spatio-temporal regions, and then "moves up" in the tree with the same goal.

The *TopDown* algorithm is conceptually more intuitive and, as we will see later, it also has a lower worst-case complexity. The spatio-temporal generalization algorithms proposed so far for location privacy follow this approach. As we detail in this section, guaranteeing the safety property by following the *BottomUp* approach is more challenging. However, considering that the counting function must be computed for any candidate region, the *BottomUp* algorithm has the advantage of being a "local" algorithm, in the sense that in many cases it terminates after processing data located in a small spatio-temporal area. Vice versa, *TopDown* always starts from the entire spatio-temporal domain and hence it requires information on all the objects.

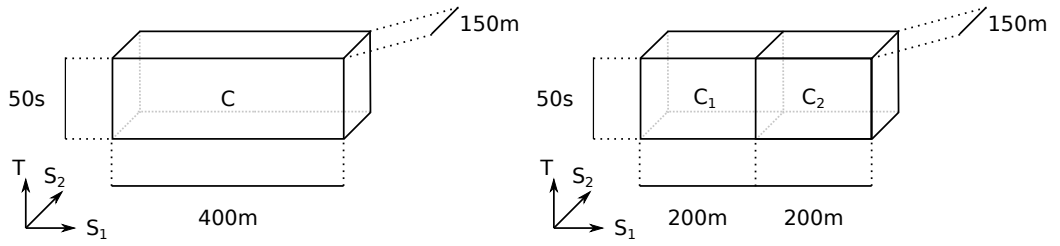
In the following of this section we first formalize the generalization tree in Section 4.1, and then we present the *TopDown* and *BottomUp* algorithms in Sections 4.2 and 4.3, respectively.

4.1 Data structure: the generalization tree

A *generalization tree* is a binary tree whose nodes represent cuboidal spatio-temporal areas that we call "spatio-temporal cells" (or "ST-cell", for short). The height of the tree is a

system parameter and the ST-cell associated with the root is the entire spatial and temporal domain (i.e., $S \times T$). The ST-cell of each non-leaf node is partitioned by the ST-cells of its two children as detailed in the following. Given this construction, it is easily seen that the nodes at a given level partition $S \times T$.

We now specify how to construct the two children $\{c_1, c_2\}$ of any internal node c in the generalization tree. Intuitively, we split c along one of the three dimensions, dividing it in two even parts. Since the two resulting cells c_1 and c_2 partition c, their projection on the other two dimensions is the same as in c, as shown in Figure 2.



(a) The ST-cell c (b) The two children of c

Figure 2: A ST-cell c and its two children c_1 and c_2

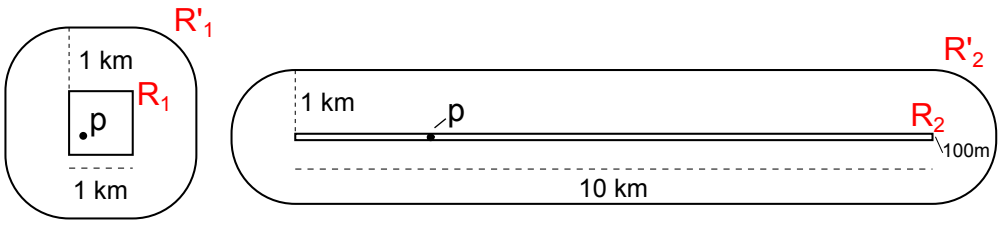
In order to decide along which dimension a cell should be divided, we follow the intuitive goal of preferring squared ST-cells over stretched ones. We explain this intuition with Example 13.

Example 13. Alice is using a fiend-finder service to be notified when one of her friends is closer than 1km. The service is "privacy-aware" and it is designed to receive generalized locations from the users.

To protect Alice's privacy, her client always generalizes Alice's location to a region R containing 10 shops. The server replies with the set of friends closer than 1km to any point of R. Finally the client filters out those friends whose position is not actually closer than 1km from the exact Alice's location.

Suppose that, for a give source position p, there are two different generalization algorithms: one returns a squared region R_1, the other the stretched region R_2 (see Figures 3(a) and 3(b)). Note that the two regions have the same area of 1km^2. Upon receiving R_1 and R_2, the service provider would return the friends in the regions R'_1 and R'_2, respectively, with R'_2 being more than 3 times larger than R'_1. Consequently we can expect, on average, that the generalization to R_2 incurs into larger communication costs and larger computational costs both on the client and on the server.

According to the above intuition (i.e., to prefer squared ST-cells), considering the two spatial dimensions we prefer to divide the cell along the dimension for which the cell has a larger size, as done in Figure 2. More application-oriented criteria are needed to decide when to generalize along the temporal dimension as opposed to the spatial ones. For this purpose, we introduce a system parameter α, called *time influence* parameter, whose value will impact the splitting decision between temporal and spatial dimensions. Intuitively, a lower value of α will privilege splitting along the spatial dimensions, while a larger value will privilege splitting along the temporal dimension. The proper value is tuned experimentally based on specific application requirements.



(a) Squared. (b) Stretched.

Figure 3: Comparison between a squared generalized region and a stretched one.

In the following definition, we use the notation $|c|_D$ to denote the projection of a ST-cell c along dimension D.

Definition 14. The *split* dimension of an ST-cell c with time influence parameter $\alpha \in [0; +\infty)$, denoted with $split_\alpha(c)$ is defined as:

$$split_\alpha(c) = \begin{cases} S_1 & \text{if } |c|_{S_1} \geq |c|_{S_2} \text{ and } |c|_{S_1} \geq \alpha \cdot |c|_T \\ S_2 & \text{if } |c|_{S_2} > |c|_{S_1} \text{ and } |c|_{S_2} \geq \alpha \cdot |c|_T \\ T & \text{otherwise} \end{cases}$$

Finally, we formalize how children are constructed.

Definition 15. Let c be an ST-cell, $D_1 = split_\alpha(c)$ be the split dimension of c, D_2 and D_3 the other two dimensions, $|c|_{D_1} = [min, max)$ and $med = \frac{max+min}{2}$. Then, the function $children_\alpha(c)$ returns the two children c_1 and c_2 of c defined as follows:

$$\begin{aligned} |c_1|_{D_1} &= [min, med) \\ |c_2|_{D_1} &= [med, max) \\ |c_1|_{D_2} &= |c_2|_{D_2} = |c|_{D_2} \\ |c_1|_{D_3} &= |c_2|_{D_3} = |c|_{D_3} \end{aligned}$$

In the following, to shorten the notations, given a ST-cell c we denote with $sib_\alpha(c)$ its sibling, and with $par_\alpha(c)$ its parent. In these notations we omit α when no confusion arises.

4.2 The *TopDown* algorithm

The intuitive idea behind the *TopDown* algorithm is to traverse the generalization tree from the root towards the leaf node that contains the source point. The algorithm terminates when it reaches that leaf node or an internal node c such that the counting function of one the children of c yields a value smaller than k. As shown in the proof of Theorem 20, this termination condition makes this generalization function safe. The basic version of the algorithm is shown in Algorithm 1.

Example 16. Consider a generalization tree like the one in Figure 4. The number reported in each leaf ST-cell indicates the value of Ω for that ST-cell. Also, consider a source point p in ST-cell C_7 (shaded in grey) and the parameter $k = 5$.

Algorithm 1 *TopDownBasic*$(p, \alpha, \Omega(), k)$

Input: a source point $p \in S \times T$, the time influence parameter α, the $\Omega()$ function, the integer value k.
Output: a SafeBox containing p or **fail**
Procedure:

1: $c = S \times T$
2: **if** $(\Omega(c) < k)$ **then return fail**
3: $\{c_1, c_2\} = children_\alpha(c)$
4: **while** $(\Omega(c_1) \geq k$ **and** $\Omega(c_2) \geq k)$ **do**
5: **if** $(p \in c_1)$ **then** $c = c_1$ **else** $c = c_2$
6: **if** $(c \in \perp)$ **return** c
7: $\{c_1, c_2\} = children_\alpha(c)$
8: **end while**
9: **return** c

TopDownBasic first considers the root C. Since the value of $\Omega(c)$ for the whole domain is equal to 12 (the sum of the counting function among all leaf ST-cells), the algorithm does not terminates here with failure (see Line 2) but instead computes the two children of C (Line 3). The condition for entering in the loop (see Line 4) is then considered: since ST-cell C_2 is such that $\Omega(C_2) < 4$, then the algorithm does not enter the loop and C is returned in Line 9 as the SafeBox.

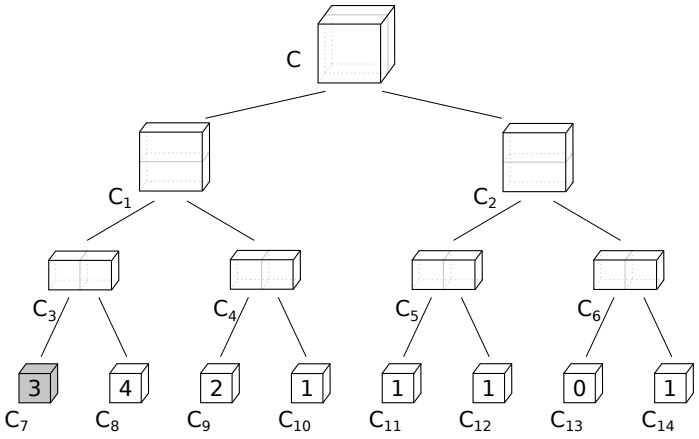


Figure 4: Example of generalization tree

Note that *TopDownBasic* returns **fail** only when k is larger than the total number of objects to count in the entire spatio-temporal domain. Indeed, in this case it is impossible to find a SafeBox, independently from the generalization function. Vice versa, in all other cases (i.e., when $\Omega(S \times T) \geq k$), *TopDownBasic* can always find a SafeBox.

Algorithm 1 has a computational issue: each time the algorithm moves one level down in the generalization tree from a ST-cell c, it needs to recompute Ω over the two children of c. While the burden of this operation can be limited by using some caching technique, we propose the optimized version *TopDown* that returns the same result of *TopDownBasic* (see Theorem 21) but that avoids computing Ω for two overlapping regions.

Algorithm 2 shows the pseudocode for *TopDown*. Variable c represent the current ST-cell being processed that is set to the entire spatio-temporal domain in Line 1. Then the algorithms enters in a **while** loop that traverses the generalization tree towards the leaf ST-cell containing the source point p. At each iteration, unless the algorithm terminates at that iteration, it only evaluates Ω for the child c_2 of c that does not contain p (Line 9). Indeed, if $\Omega(c_2) \geq k$, the algorithm does not *directly* check if $\Omega(c_1) \geq k$, where c_1 is the child of c containing p. Instead, the algorithm processes c_1 in the following iteration. If the Ω function applied to a child of c_1 yields a value not smaller than k, then, due to the monotonic property of Ω (see Definition 2) $\Omega(c_1) \geq k$. In practice, with this approach in most of the cases (i.e., all the cases in which the algorithm continues in the iteration) we have an *indirect* evaluation of the condition $\Omega(c_1) \geq k$.

When $\Omega(c_2) < k$ we cannot indirectly infer if $\Omega(c) \geq k$ and the algorithm explicitly needs to compute this condition (see Lines 11 to 15). If $\Omega(c) \geq k$ then the result is c itself. Otherwise, due to the termination condition of *TopDownBasic*, the result is the parent of c. In case c is the entire spatio-temporal domain (i.e., the root of the generalization tree), the algorithm returns **fail**. This happens only when the Ω function applied to the entire spatio-temporal domain yields a values smaller than k.

Finally, there is another termination condition: when the algorithm reaches a leaf ST-cell c (see Lines 3 to 6). Also in this case it is not possible to indirectly evaluate if $\Omega(c) \geq k$, hence the algorithm explicitly compute this condition and it returns c if the condition is met, the parent of c otherwise.

Algorithm 2 *TopDown*

Input: a source point $p \in S \times T$, the time influence parameter α, the $\Omega()$ function, the integer value k.
Output: a SafeBox containing p or **fail**
Procedure *TopDown*(p, α, Ω, k)

1: $c = S \times T$;
2: **while true do**
3: **if** $(c \in \perp)$ **then**
4: **if** $(\Omega(c) \geq k)$ **then return** c
5: **else then return** $parent_\alpha(c)$
6: **end if**
7: c_1 is the ST-cell in $children_\alpha(c)$ such that $p \in c_1$
8: c_2 is the ST-cell in $children_\alpha(c)$ such that $p \notin c_2$
9: **if** $(\Omega(c_2) \geq k)$ **then**
10: $c = c_1$
11: **else**
12: **if**$(\Omega(c) \geq k)$ **then return** c
13: **else if** $(c = S \times T)$ **then return fail**
14: **else return** $parent_\alpha(c)$
15: **end if**
16: **end while**

4.3 The *BottomUp* algorithm

The *BottomUp* algorithm processes the generalization tree from the leaf node containing the source point towards the root.

An intuitive procedure would first identify the leaf node containing the source point p and recursively move to the parent ST-cell in case the value of Ω for that ST-cell is less than k, and returning the current node as the SafeBox when Ω is at least k. We proposed a similar algorithm, called $\mathcal{I}ncognitus$, in a preliminary investigation on this problem [4]. Unfortunately, the result obtained with this approach is indeed a *SafeBox*, but the generalization function is not safe according to our definition, as shown in Example 17.

Example 17. Consider the generalization tree reported in Figure 5, a value of k equal to 4 and a bottom up generalization algorithm that, starting from the leaf ST-cell containing the source point, continues to generalize if the value of Ω for the current ST-cell is less than k.

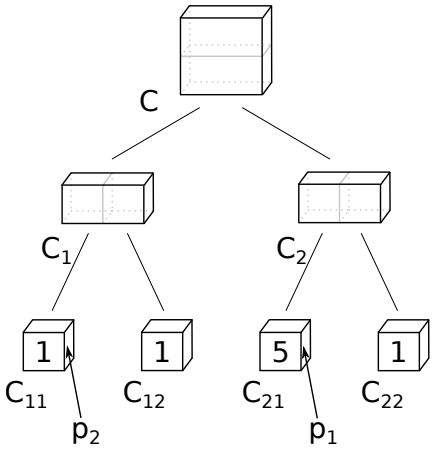


Figure 5: Unsafety of an intuitive bottom-up strategy

The generalization of p_1 is C_{21}, since $\Omega(C_{21}) = 5 \geq k$. If the source point is p_2, the algorithm first processes C_{11} but discards this as a SafeBox, because $\Omega(C_{11}) = 1 < k$. Then the algorithm moves up in the tree, processing ST-cell C_1. This is not a suitable SafeBox neither, since $\Omega(C_1) = 2 < k$. Then the algorithm moves up to c that is a SafeBox since $\Omega(C) = 8 \geq k$.

This algorithm has the same problem illustrated in Example 4. Intuitively, by observing a generalization to C, an adversary may exclude as a candidate source point any point in C_{21} (since it would be generalized only to C_{21}). Hence the privacy preference of having at least k objects around the source point would be violated. Technically this is captured by the fact that $I(C_0) = C_{11} \cup C_{12} \cup C_{22}$. Since $\Omega(C_{11} \cup C_{12} \cup C_{22}) < k$, this algorithm is not safe.

The *BottomUp* algorithm we propose in this paper fixes the above problem by adopting a more involved strategy and a different termination condition. The general idea is that, instead of counting the number of objects in the current node c, we count the number of objects in the inversion of c, i.e., in the ST-area defined as the set of all potential source points. The algorithm returns c only if the counting function Ω applied to this area returns a value not less than k. While the idea is quite simple, the computation of the inversion is not, as we will illustrate below.

We first formally describe the *BottomUp* algorithm, and then we provide an example of its application. Algorithm 3 starts from the leaf ST-cell containing the source point p (Line 1) that is also assigned to variable c used in the main loop. If Ω for that ST-cell is greater than or equal to k, the algorithm returns that ST-cell, since for leaf nodes $I(c) = c$ if $\Omega(c) \geq k$ (Line 8). If this is not the case, the algorithm enters in the **while** loop that traverses the generalization tree towards the root. If the algorithm actually reaches the root without finding a node that satisfies the termination condition, then it fails since it is not possible to obtain a SafeBox for the given source point with the considered procedure (Line 4). Note that this implies that in some cases *BottomUp* could not be able to find a SafeBox while a different algorithm (e.g., *TopDown*) actually could. However, this a very rare situation (see Section 5).

Vice versa, if the considered ST-cell ($currentSTcell$) is an internal node, its value is updated with the one of its parent node in the generalization tree (Line 5), and the counting function is evaluated on the inversion of this new ST-cell, computed through the *BottomUpInversion* function (presented in the following).

Algorithm 3 *BottomUp*

Input: a source point p, a time influence parameter α, the $\Omega()$ function, the integer value k.
Output: a SafeBox containing p or **fail**
Procedure:

1: $currentSTcell$ is the leaf ST-cell such that $p \in currentSTcell$
2: $c = currentSTcell$
3: **while** ($\Omega(c) < k$) **do**
4: **if** ($currentSTcell = S \times T$) **then return fail**
5: $currentSTcell = par_\alpha(currentSTcell)$
6: $c =$ *BottomUpInversion*$(currentSTcell, \alpha, \Omega, k)$
7: **end while**
8: **return** $currentSTcell$

Example 18. Let's consider again the generalization tree in Figure 4. As in Example 16 the goal is to find the SafeBox containing at least 4 objects. The *BottomUp* algorithm starts computing the leaf ST-cell containing p in Line 1 (identifying the ST-cell C_7 shaded in grey in Figure 4). Then it computes the value $\Omega(C_7) = 3$; since the requirement of having at least 4 objects is not fulfilled, the algorithm enters the while loop (Line 3). Since $C_7 \neq S \times T$, the algorithm computes the parent of C_7 in Line 5, considering as candidate SafeBox the ST-cell C_3 in Figure 4. The inversion of C_3, computed by the *BottomUpInversion* procedure (Line 6), is the empty set: indeed, C_3 is the union of C_7 and C_8, and any point in C_8 would have as SafeBox C_8 itself because its $\Omega()$ is equal to 4. Furthermore any point in C_7 will have as SafeBox an area greater than the one represented by C_3, because otherwise the only candidate source points for the generalization to C_3 will be the one in C_7 and the counting in that area is insufficient to guarantee the user privacy preferences. Since the counting for an empty set is zero, the algorithm enters again in the loop. Since $C_3 \neq S \times T$ the parent of C_3 is computed as C_1. The inversion of C_1 is the union of C_7, C_9, and C_{10} since both the Ω value of C_9 and C_{10} and the Ω of their union C_4 are smaller than 4, and hence, any point in those ST-cells would have as SafeBox the region corresponding to the node C_1 or to an ancestor of C_1. It is indeed C_1 because the counting function gives a value higher than 4 for the region corresponding to the union of C_7, C_9, and C_{10}. In the evaluation of the loop condition we have $\Omega(\{C_7, C_9, C_{10}\}) = 6$ and hence the comparison with $k = 4$ leads the

algorithm to exit the loop, returning C_1 as the SafeBox.

Note that in Example 18 we just gave an intuitive motivation for the result of the inversion computation. Two problems arise when trying to directly apply Definition 5 to compute $I(c)$ for a ST-cell c when the generalization function is *BottomUp*. First, according to the definition, it would be necessary to compute *BottomUp* for each point in c, which is impossible if we consider a continuous spatio-temporal domain and impractical even assuming a discrete domain. Second, this would generate a non-terminating procedure. Indeed, according to Definition 5, for any $p \in c$ we need to compute *BottomUp* with p as a source point. However, the computation of *BottomUp* requires computing the inversion for an area that contains p, which is clearly an endless recursion.

The first problem can be easily fixed. Indeed, in *BottomUp* all points belonging to the same leaf node are generalized to the same SafeBox. So, instead of checking the inversion property for each point in the current candidate SafeBox, we can choose a representative point for each leaf ST-cell in the candidate SafeBox and check the condition for these points only.

To solve the second problem (non termination), we adopt Procedure 4. The idea is that, instead of directly computing Definition 5, which applies to any generalization function, we adopt Procedure 4 that is specific for *BottomUp* and that, in the computation, does not require to recursively call *BottomUp* itself.

Procedure 4 *BottomUpInversion*

Input: a ST-cell c, a time influence parameter α, the $\Omega()$ function, the integer value k.
Output: $I_{BottomUp}(c)$
Procedure *BottomUpInversion*(c, α, Ω, k)

1: $res =$ *Residuals*(c, α, Ω, k);
2: **if**$(\Omega(res) \geq k)$ **then return** res
3: **else return** $\emptyset$

Procedure *Residuals***(**c, α, Ω, k**)**

1: **if** (c is leaf) **then return** c
2: $result = \emptyset$
3: **for each** c' in $children_\alpha(c)$ **do**
4: $\quad A =$ *Residuals*(c', α, Ω, k)
5: $\quad$ **if** $(\Omega(A) < k)$ **then** $result = result \cup A$
6: **end for**
7: **return** $result$

Our solution to compute $I(c)$ consists in the *BottomUpInversion* procedure (see Procedure 4) that uses *Residuals*, a recursive procedure that computes the set of all points whose generalization is not smaller than c. If the counting function applied to this set is larger than or equal to k, then this set is returned (Line 2). Indeed, all of these points (res) generalize to c. Otherwise, (i.e., $\Omega(res) < k$), the counting condition in *BottomUp* for the points in res is not satisfied and hence the generalization of each of these points is larger than c. In this case, $\emptyset$ is returned as required by the definition of $I(c)$.

Procedure *Residuals* processes every node of the subtree with root in c. Recursion terminates when *Residuals* is called on a leaf node c. In this case, the leaf node itself is returned. Indeed, for each point in c, the result of the generalization is at least as large as c itself. For an internal node c, *Residuals* recursively calls itself on the two children of c. For each child

c' of c, *Residuals* checks if the counting function applied to *Residuals*(c', α, Ω, k) is less than k. If this is the case, then every point in the result of *Residuals*(c', α, Ω, k) is added to the result that is being computed for c (Line 5). Otherwise, (i.e., Ω(*Residuals*$(c', \alpha, \Omega, k)) \geq k$) every point in c' generalizes to a ST-cell smaller than c, hence no point of c' contributes to the result.

Example 19. Let's consider Example 18 that illustrates the application of the *BottomUp* algorithm to the generalization tree in Figure 4, and in particular to a source point p in the ST-cell C_7. The algorithm requires the computation of *BottomUpInversion*(C_3) and of *BottomUpInversion*(C_1) that we intuitively motivated as equal to $\emptyset$ and to $C_7 \cup C_9 \cup C_{10}$, respectively. Consider first *BottomUpInversion*(C_3) [1]. The *BottomUpInversion* procedure, first computes the set res through the *Residuals* procedure. Since C_3 is not a leaf, the inner iteration of *Residuals* considers first $c' = C_7$ and then $c' = C_8$. For $c' = C_7$, $A = \{C_7\}$, and since $\Omega(A)$ is less than 4, C_7 is added to $result$. For $c' = C_8$, $A = \{C_8\}$, and since $\Omega(A)$ is greater than 4, $result$ is not modified. The *Residuals* procedure returns the set containing only C_7, so we have that $res = \{C_7\}$. Since $\Omega(res) < 4$, the *BottomUpInversion* procedure terminates returning the empty set.

Consider now the computation of *BottomUpInversion*(C_1). Since C_1 is not a leaf, the inner iteration of *Residuals* considers first $c' = C_3$ and then $c' = C_4$. When *Residuals* considers C_3, as we have seen before, it returns $result = \{C_7\}$. Then it considers C_4 and computes $A = C_9 \cup C_{10}$ because both of these cells are leaves and the counting function applied to each of them is less than 4. Since $\Omega(A) = 3$ is less than 4, the procedure returns $C_9 \cup C_{10}$. Consequently, *Residuals* applied to C_1 returns $result = C_7 \cup C_9 \cup C_{10}$. Hence in *BottomUpInversion*(C_1) we have $res = C_7 \cup C_9 \cup C_{10}$ and since $\Omega(res) = 6$, the procedure returns $C_7 \cup C_9 \cup C_{10}$, as expected from our intuitive reasoning in Example 18.

4.4 Properties of the *TopDown* and *BottomUp* algorithms

In this subsection we consider the formal properties of the two algorithms that we have presented and we compare their worst-case time complexity.

4.4.1 Safety

In order to show the correctness of the algorithms we have to prove that both *TopDown* and *BottomUp* compute a safe generalization function. For *TopDown*, we first show that *TopDownBasic* is a safe generalization function and then we show that *TopDown* computes the same result as *TopDownBasic* (hence *TopDown*is a safe generalization function). This is formally stated in Theorems 20, 21.

Theorem 20. *The* TopDownBasic *algorithm computes a safe generalization function.*

Theorem 21. *For any source point p, any time influence parameter α, any $\Omega()$ function, and any the integer value k it holds that* TopDownBasic(p, α, Ω, k) = TopDown(p, α, Ω, k).

Before presenting the formal result for *BottomUp* in Theorem 22, we first present Property 4 that formally states that *BottomUpInversion* actually computes the inversion for *BottomUp*.

Property 4. Let $\mathcal{G}$ be the generalization function computed by *BottomUp* with time influence parameter α, counting function $\Omega()$, and integer value k. For each ST-cell c, *BottomUpInversion*(c, α, $\Omega()$, k) computes $I_{\mathcal{G}}(c)$.

[1] For simplicity, we omit the other parameters of the procedure.

Theorem 22. *The* BottomUp *algorithm computes a safe generalization function.*

The formal proofs of the above theorems are reported in Appendix A. Intuitively, the idea of both proofs is the following: we first show that if the algorithm does not return **fail**, it returns a ST-cell c that contains the source point p (this guarantees that the algorithm actually computes a generalization function) and then, according to Definition 6, we prove that $\Omega(I(c))$ is not smaller than k.

4.4.2 Analysis of computational complexity

We first consider the worst-case time complexity. For each iteration of the main loop, the only operation in *TopDown* that does not require a constant time is the computation of $\Omega()$. In the worst-case (i.e., when the algorithm returns a leaf node) *TopDown* computes $\Omega()$ once for each level of the tree. Hence the worst-case time complexity of *TopDown* is linear in the height of the generalization tree times the complexity of computing Ω.

Before analyzing the complexity of *BottomUp*, we first introduce a simple but effective optimization. In principle the computation of *BottomUpInversion*(c), as described in Procedure 4, would require to compute Ω for each node in the subtree with root c. However, by definition of *BottomUp*, if c is an internal node, we compute *BottomUpInversion*(c) only if we have already computed *BottomUpInversion*(c') for one child c' of c. By storing the result of *BottomUpInversion*(c'), we avoid to re-process the subtree with root in c' when computing *BottomUpInversion*(c). With this optimization, we never compute $\Omega(c)$ twice for the same ST-cell c. Consequently, in the worst-case (i.e., when *BottomUp* traverses the generalization tree up to the root), we need to compute Ω for each node in the tree.

Comparing the two algorithms, given a generalization tree of height h, *TopDown* requires computing Ω a number of times linear in h, since *TopDown* "moves down" in the generalization tree at each iteration. Vice versa, in the worst case *BottomUp* needs to process all nodes of the generalization tree, hence it is linear in the number of nodes (i.e., 2^h) and, consequently, exponential in h. We recall that, by definition of the generalization tree (see Section 4.1), the height of the tree (and hence the number of its node) is a system parameter. The choice of a value for h is subject to a trade-off. On one side, for higher generalization trees we get smaller bottom ST-cell and this positively impacts on the average size of the generalization regions (this strongly affects *BottomUp* and, minimally *TopDown*). On the other side, for larger values of h we have a higher computation time (again, this affects *BottomUp* significantly and and *TopDown* only minimally). In Section 5 we show the impact of this parameter in our experimental setting.

Let's now consider the worst-case time complexity of the two algorithms by also taking into account the complexity of computing Ω. Clearly, the complexity of Ω depends on the data structure used to store the objects. In our experiments we use two different data structures. As we illustrate in Section 5, by using one of them we have a worst-case time complexity of Ω linear in the number of leaf ST-cells contained in the considered area. With this data structure, we can evaluate the complexity of the two algorithms in terms of total number of leaf ST-cells that each algorithm needs to process in all the computations of Ω. According to this metric, given h the height of the generalization tree, the worst-case time complexity of *TopDown* is $O(2^{h+1})$. Indeed, in the first iteration *TopDown* computes Ω on the entire spatio-temporal domain (i.e., 2^h leaf ST-cells), in the second iteration it computes omega on half of the spatio-temporal domain (i.e., 2^{h-1} leaf ST-cells) and so on. Consequently, in the worst-case, the number of processed ST-cells is $\sum_{i=0}^{h} 2^h = 2^{h+1} - 1$. The worst-case time complexity of *BottomUp* is $O(h \cdot 2^h)$. Indeed, in the worst-case *BottomUp*

computes omega for each node of the generalization tree. Since, the union of all the nodes at the same height yields the entire spatio-temporal domain (i.e., 2^h leaf ST-cells), the total number of ST-cells to process is equal to $h \cdot 2^h$. As a result, comparing the worst-case time complexity of the two algorithms, *BottomUp* is expected to be only $h/2$ times slower than *TopDown*.

5 Experimental evaluation

In this section we evaluate the SafeBox computation algorithms described in Section 4. Using two different datasets, we first evaluate the effectiveness of *TopDown* and *BottomUp* algorithms by showing how parameter k impacts on the data quality, i.e., the spatial size and temporal duration of SafeBoxes. Intuitively, as long as a spatio-temporal region is a SafeBox, it should be as small as possible so as the approximation involved in its use is reduced. Secondly, we empirically evaluate the performance of the two algorithms in terms of computation time. Finally we compare *TopDown* and *BottomUp* with our previous solution *Grid* [12].

5.1 Experimental setting

The spatial area in which the experiments have been conducted is the city of Milano and the total size of the map is $325\ km^2$. In our experiments we use two datasets: the first represents the movement of a set of users, as described in Section 5.1.1, while the second dataset includes all the shops in Milano with opening hours (see Section 5.1.2). Hence the "objects" counted by $\Omega()$ will be users and shops, respectively. The choice of using different datasets is aimed at testing our algorithms with both dynamic and relatively static data. The results are computed as the average, as well as minimum and maximum, out of 1000 runs. In each run, a random source point is chosen. The parameter k represents the minimum number of objects that each SafeBox returned by the algorithms should contain.

The α value, as described in Section 4.2, determines different ratios between the spatial and temporal sizes of the SafeBox. The choice of α is strictly related to the application we are using: if we are in a real time environment we need to keep the temporal size, and hence temporal obfuscation, as small as possible, while in other contexts it could be useful to have smaller spatial areas or balance the space and time components. In all the experiments presented in this section we use a value of $\alpha = 1.6 \times 10^{-5}$ such that $\alpha \cdot 60s = 0.787km$: this value, chosen empirically, produces SafeBoxes with a duration that is, on average, under 15 minutes. This is a reasonable length if, for instance, we think about a service that uses the position for sharing purposes.

Other parameters that should be taken into account are the subdivisions along the two spatial dimensions and the temporal dimension. The finer is the subdivision, the smaller are the resulting leaf ST-cells: this implies that the height of the corresponding generalization tree is higher, which negatively impacts on the computation time. The number of leaf ST-cells affects both the computational complexity, as pointed out in Section 4, and the precision of the algorithms. We vary the number of leaf ST-cells from $2,048$ to $8,388,608$. This yields to a spatial subdivision that varies from about 1.1km to 70m and a temporal one from 75 minutes to 2 minutes. Details of the subdivisions are shown in Table 1 with default values, as used in some experiments, written in bold. The spatial projection of each leaf ST-cell is square-shaped.

The two algorithms have been implemented in Java. Geographical positions are represented as latitude and longitude values in decimal degrees. The experiments have been conducted is on computer with 2.5GHz Intel i5 CPU with 4GB of main memory.

Subdivision	**Area (Km^2)**	**Duration (min)**	
		MilanoByNight	Google Shops
Coarsest	1.28	45	75
Coarse	0.32	22	37
Mid	0.08	11	18
Fine	**0.02**	**5**	**9**
Finest	0.005	2	4

Table 1: Spatial and temporal leaf ST-cell size

5.1.1 MilanoByNight

MilanoByNight is an artificial dataset of user movements obtained using a simulation that reflects a typical scenario of a weekend night in the city of Milano. It includes $100,000$ potential users, moving to one or more entertainment places in a period of 6 hours[2].

The average density of the users within this area is $465\ users/km^2$. In this scenario, given a ST-cell c, Ω counts the number of users within c, according to the appearance semantics described in Section 3.3. In Table 2 the values of k are summarized.

Parameter	**Values**
k	20, 40, 60, 80, **100**, 120, 140, 160

Table 2: MilanoByNight k values

5.1.2 Google Shops

This dataset considers the same spatial area of MilanoByNight, but instead of considering users as the set of objects, it considers shops. The dataset includes $12,958$ shops whose position and properties were retrieved through Google Places API. An opening timetable is assigned to each shop, using real values when available (about $2,000$ shops) and assigning default opening times in the other cases. The considered temporal domain is 10 hours long, from 9.00AM until 19.00PM of a given day.

The $\Omega()$ function counts the *open* shops in candidate SafeBoxes. We test our algorithms with both the semantics described in Section 3.3: appearance and persistence. Important parameters for the experiments on this dataset are the number k of shops and the persistence interval duration D. Their considered values are reported in Table 3.

5.2 Computation of the counting function

The efficiency of both algorithms depends on how Ω is computed. Several techniques can be used to implement Ω and optimizations are possible depending on the application con-

[2] http://everywarelab.di.unimi.it/lbs-datasim

Parameter	Values
k	2, 4, 6, 8, **10**, 12, 14, 16
D (minutes)	5, 15, 30

Table 3: GoogleShops parameters values

text. If the objects that the application needs to count are relatively static in both time and space, as for example in the case of train stations, then the value of Ω for each ST-cell in the whole generalization tree can be pre-computed. This implies that given a ST-cell c, $\Omega(c)$ can be obtained in constant time, independently from c being the whole spatial and temporal domain $S \times T$ or a leaf ST-cell.

In contrast, in this experimental evaluation, we assume that the data to be counted is not static and cannot be precomputed. This assumption is reasonable for both datasets we use in our experiments. In the MilanoByNight dataset, $\Omega()$ counts users considering their location at given time instants and, hence, it cannot be precomputed because movements are unpredictable. The other dataset includes more than $10,000$ shops with their opening hours, and in a big city like Milano shops can frequently change their presence, location, and opening times, specially in the city center. Hence, for both datasets we assume that counting is performed by accessing an external service as opposed to querying a static internal database and we do not perform any pre-computation of $\Omega()$ for larger areas like the ones corresponding to internal nodes of the generalization tree. Indeed, for the Google shops dataset, we also test the algorithm with online retrieval of data from Google servers.

In the MilanoByNight dataset, we store, for each leaf ST-cell, the set of users reported to be in that ST-cell (both in space and time). Consequently in order to compute Ω on an area A, it is necessary to process all the leaf ST-cell contained in A.

In the Google shops dataset we use two different approaches. In the first approach (see Sections 5.3.2 and 5.3.3), we store objects in a data structure build as the spatial projection of the leaf ST-cells (we recall that objects in this dataset are not moving). In each cell of this spatial grid we store the shops whose position is within the cell, each one paired with its corresponding opening hours (stored as a list of intervals). In the second approach (see Section 5.3.4) we compute Ω by actually retrieving shops information with online queries to Google servers.

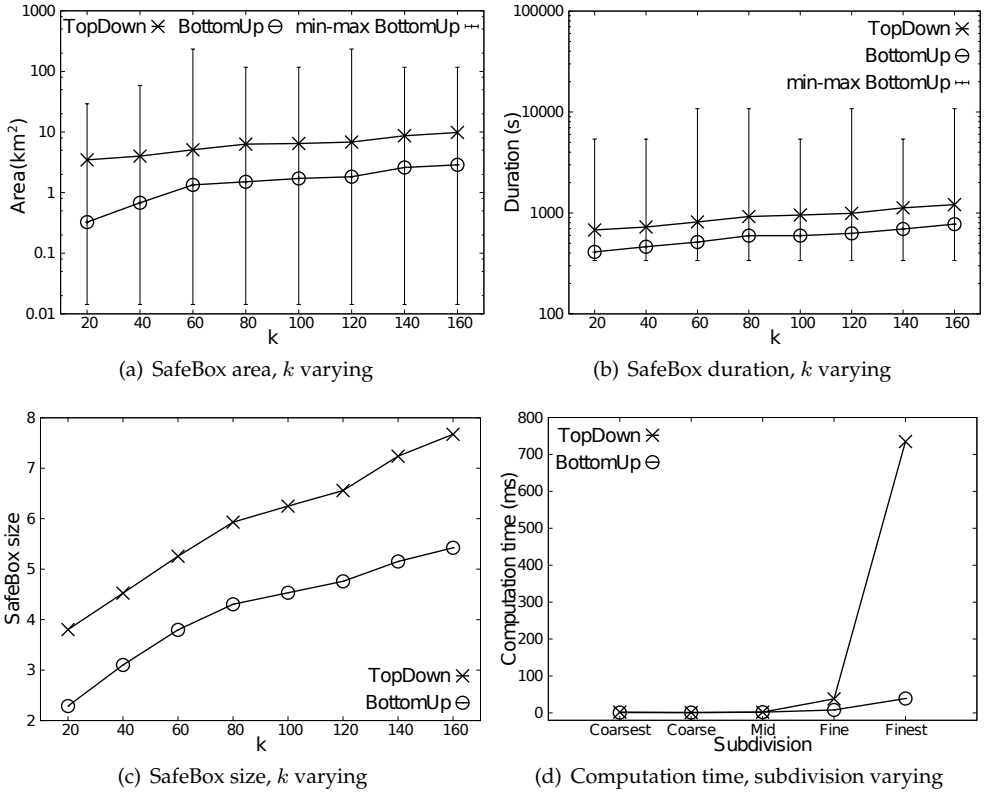
5.3 Evaluation

In this section we analyze and discuss the experimental results. In Section 5.3.1 we present the MilanoByNight results, and in Section 5.3.2 we show the results with the Google shop dataset, both adopting appearance semantics. Experimental results with persistence semantics using the Google shop dataset are shown in Section 5.3.3. In Section 5.3.4 we show the results with the online retrieval of the Google Shops dataset, while in the last set of experiments in Section 5.3.5 we compare the *TopDown* and *BottomUp* with the *Grid* algorithm [12].

In all the test we conducted, the percentage of **fail** result returned by *BottomUp* is less than 0.05% (i.e., $54/112,000$) while *TopDown* never returned **fail**, as expected (see Section 4.2).

5.3.1 Evaluation with MilanoByNight dataset

The first set of experiments tests the two algorithms' precision and performance with the MilanoByNight dataset adopting the appearance semantics.



(a) SafeBox area, k varying

(b) SafeBox duration, k varying

(c) SafeBox size, k varying

(d) Computation time, subdivision varying

Figure 6: Results with MilanoByNight dataset

In Figure 6(a) we compare the average size of the spatial areas of the SafeBoxes returned by *TopDown* and *BottomUp* for different values of k. As expected, by increasing the value of k, slightly larger areas are returned by both algorithms. The comparison between the two algorithms shows that on average the *BottomUp* algorithm produces much smaller areas (up to an order of magnitude) with respect to the *TopDown* algorithm. A similar result is shown in Figure 6(b), in which the duration of the SafeBoxes is compared.

In Figure 6(a), the bars represent the maximum and the minimum size of the areas returned by *BottomUp*. The area's variance for the *TopDown* algorithm is not reported since it is small compared to *BottomUp* one. The reason of a high variance for *BottomUp* can find an explanation in the generalization strategy used by the algorithm. Indeed, the algorithm takes decisions based on local conditions starting from the leaf nodes. A non-uniform distribution of objects in the considered space leads to significantly different decisions depending on the position of the source point leading to significantly different generalizations.

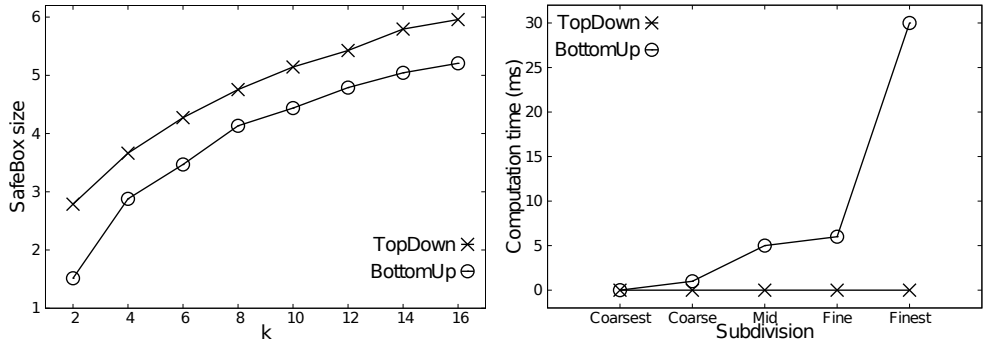
The overall precision, in both time and space, is summarized in Figure 6(c) that compares *TopDown* and *BottomUp* in terms of the "size" of the returned SafeBoxes: technically the size

of a SafeBox is the level of the SafeBox in the generalization tree described in Section 4.1. This implies that lower is the value, the smaller is the SafeBox in both time and space, and hence the higher is the precision. Figure 6(c) confirms the experimental results of area and duration comparisons: the *BottomUp* algorithm produces smaller SafeBoxes up to 3 levels in the generalization tree, meaning that a *TopDown* SafeBox can be $2^3 = 8$ times bigger than a *BottomUp* SafeBox.

Figure 6(d) shows how the subdivision impacts on the performance of both algorithms. We can observe that the execution time of *BottomUp* algorithm is only slightly affected by the increase of the number of leaf ST-cells, while the *TopDown* performance is quite related with it. The execution time is under 100 milliseconds in the coarsest subdivision and more than 700 milliseconds in the finest subdivision: this last value can be incompatible with a service that needs to return SafeBoxes in real time.

5.3.2 Evaluation with Google shop dataset with appearance semantics

The second set of experiments tests the two algorithms precision and performance with the Google shops dataset with appearance semantics.



(a) SafeBox size, k varying (b) Computation time, subdivision varying

Figure 7: Results with Google Shops dataset - Appearance semantics

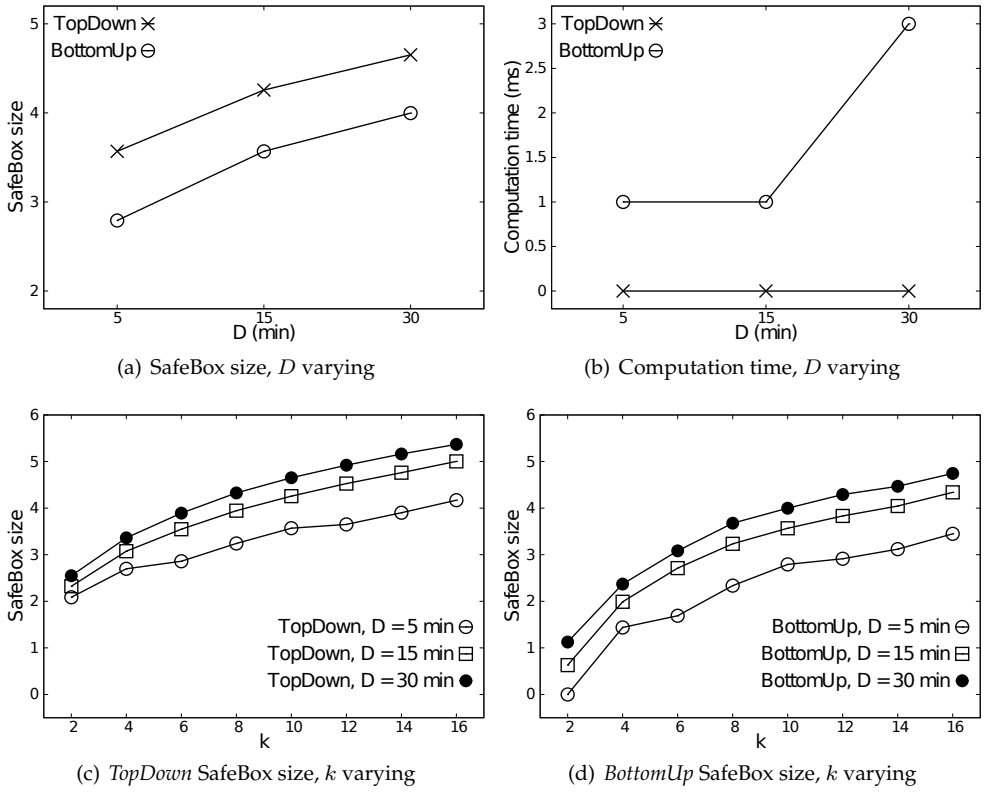
Figure 7(a) shows the size of the SafeBoxes returned by *TopDown* and *BottomUp*. By increasing the value of k, larger areas are returned by both algorithms. The SafeBox returned by *BottomUp* is on average smaller in size than the one returned by *TopDown*, as shown in Figure 7(a). Therefore, we can conclude that the *BottomUp* algorithm is preferable since it produces on average smaller SafeBox.

Comparing this result with the one of the MilanoByNight dataset we can observe that the average size of SafeBoxes returned by *TopDown* with the Google shops dataset is smaller in comparison with the one returned with MilanoByNight; conversely the average size of SafeBoxes returned by *BottomUp* is quite the same. The reason is that the distribution of users is less uniform than the distribution of the shops and this negatively affects the performance of *TopDown*. Vice versa *BottomUp* is not significantly affected by the distribution of the objects in the space.

The execution times are shown in Figure 7(b): we can observe an opposite trend with respect to the MilanoByNight dataset. The computation time of *TopDown* is always less than one millisecond while the execution time of the *BottomUp* algorithm is slightly affected by the increase of the number of leaf ST-cells. Indeed we can observe that with the finest

subdivision *BottomUp* takes up to 30 milliseconds. The difference with respect to the MilanoByNight dataset is due to the different implementation of the Ω function. In the Google shops setting, *TopDown* performs better since the computation of $\Omega()$ is very efficient, while in *BottomUp* the computation time is affected by the computation of the *Residuals* function. In any case, the computation time, even with the *BottomUp* algorithm, is less than 30 milliseconds, hence granting a quick response time.

5.3.3 Evaluation with Google shop dataset with persistence semantics



(a) SafeBox size, D varying

(b) Computation time, D varying

(c) *TopDown* SafeBox size, k varying

(d) *BottomUp* SafeBox size, k varying

Figure 8: Results with Google Shops dataset - Persistence semantics

This set of experiments evaluates the two algorithms precision and performance with the Google shops dataset adopting the persistence semantics. Figure 8(a) shows the average SafeBox size of *TopDown* and *BottomUp* algorithms when varying the persistence interval duration D from 5 minutes to 30 minutes. As expected, for shorter intervals the SafeBoxes are smaller, on average. This is due to the fact that, for a given ST-cell c, if D is short, the objects spatially contained in c are more likely to be counted more times in the computation of $\Omega(c)$. Clearly larger values of Ω result, for both algorithms, in smaller SafeBoxes.

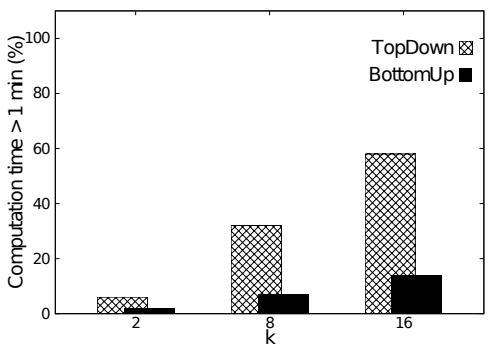
Figure 8(b) shows the computation time of the algorithms varying the persistence interval duration. As also shown in Figure 7(b), the computation time of *TopDown* is not affected by the increase of the persistence interval duration. Conversely, *BottomUp* algorithm shows a small increase of the computation with the 30 minutes persistence interval duration. This

is due to the fact that longer persistence interval duration leads to larger SafeBoxes whose computation with *BottomUp* requires more iterations of the algorithm and hence an increase in the computation time (affected in particular by the *Residuals* execution time).

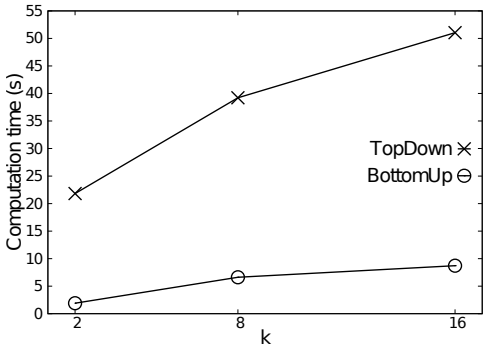
Figure 8(c) shows the SafeBox size by varying k with different value of persistence interval duration D for the *TopDown* algorithm. The average size of the SafeBoxes grows with larger value of k and longer persistence interval durations, confirming both the results shown in Figure 7(a) and in Figure 8(a). Figure 8(d) shows a similar result for *BottomUp*.

5.3.4 Evaluation with Google shop dataset, appearance semantics and online retrieval of data

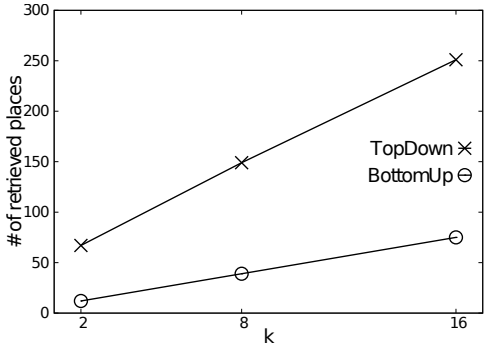
Similarly to Section 5.3.2, in this set of experiments we evaluate the performance of *TopDown* and *BottomUp* algorithms with the Google Shops dataset adopting the appearance semantics and varying k. In this set of experiments when data is needed to compute Ω we retrieve it from Google servers through the Google Places API. Clearly, in this set of experiments the generalization algorithms need to issue a (possibly large) number of requests and this leads, in some cases, to much longer computations. In this set of experiments we give a limit of one minute for each generalization. If the generalization algorithm does not terminate within this time, the generalization procedure is interrupted and it is considered "timed-out".



(a) Percentage of failed experiments, k varying



(b) Computation time, k varying



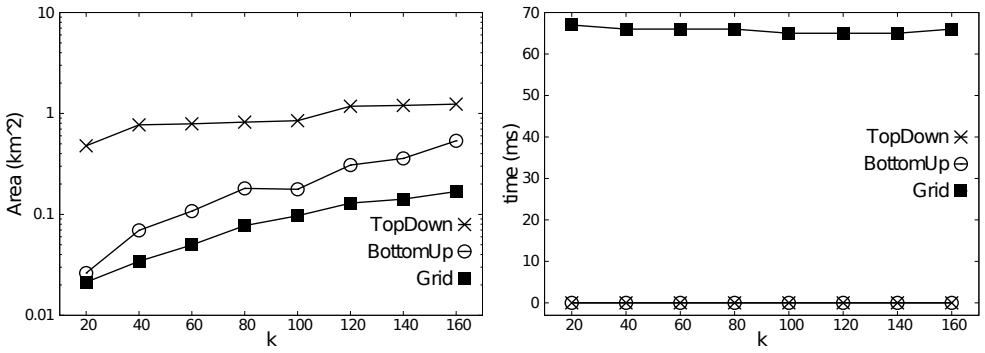
(c) Number of Places, k varying

Figure 9: Results with online retrieval of Google Shops dataset

Figure 9(a) shows the percentage of runs that were timed-out for our proposed algorithms. We can observe that for $k = 2$ this percentage is of about 2% and 6%, for *BottomUp* and *TopDown*, respectively. When increasing the value of k up to 16 the percentage grows, since both algorithms need to retrieve more points (see Figure 9(c)). However, while with *TopDown* more than 60% of the generalization runs are timed out with $k = 16$, with *BottomUp* the percentage is much smaller (i.e., less than 10%).

Figure 9(b) shows the average computation time computed among the runs that were not timed-out. We can observe that *TopDown* has a much larger computation time than *BottomUp* for different values of k. The main reason for this result is that *BottomUp* requires to retrieve less points. Indeed in Figure 9(c) we can observe that, for all considered values of k, *BottomUp* requires about one fifth of the points that are needed by *TopDown*.

5.3.5 Comparison with Grid algorithm



(a) SafeBox size, k varying

(b) Computation time, k varying

Figure 10: Comparison with existing solutions

In this last set of experiments we compare the *TopDown* and *BottomUp* algorithms with the *Grid* spatial generalization algorithm, presented in [12]. These experiments have been run with the MilanoByNight dataset adopting the appearance semantics. As in the previous experiments, the spatial domain S is Milano's area, but the temporal domain consists in only a screenshot of MilanoByNight's temporal duration, since the *Grid* algorithm provides only a spatial generalization. We briefly recall that *Grid* algorithm adopts a top-down strategy, in which a total order on the data (e.g., users' locations) needs to be established for computing the generalized area.

In Figure 10(a) we can observe, for different values of k, that *Grid* returns, on average, smaller areas and hence can provide a higher data utility. However Figure 10(b) shows that *Grid* has a much higher computation time.

In our previous work [12] we compared *Grid* with the *Hilbert Cloak* algorithm presented in [11]. From the evaluation it emerges that *Hilbert Cloak* is faster than *Grid* but returns, on average, larger areas. Consequently, we can derive that *Hilbert Cloak* has about the same performance as *TopDown* and *BottomUp* for the computation time and the quality of the result. However, both *Grid* and *Hilbert Cloak* suffer from a major problem: they require to know all the points in the spatial domain. This means that, if the test were conducted with the online retrieving of data (like in Section 5.3.4), both *Grid* and *Hilbert Cloak* would always be timed-out.

6 Conclusions and future work

In this paper we addressed the problem of privacy preservation through spatio-temporal generalization, and we presented a new problem formalization that makes the generalization technique independent from the semantics of the generalized region. This approach makes it possible to define generalization algorithms that can be easily adapted to different applications and privacy preferences by simply changing the counting function Ω that defines the semantics of the generalized regions. We showed examples of application of our algorithm with two Ω functions counting different types of objects, and with two different semantics, but its application can be easily extended to capture other types of objects and different semantics. Another innovative aspect of the proposed model is that it re-defines a formal property that generalization functions should satisfy in order to be safe with respect to the inversion attack. The *BottomUp* algorithm satisfies this property and also has the advantage of being applicable also when the generalization algorithm needs to process dynamic data possibly collected from third parties (e.g., Google Maps). We experimentally showed that this algorithm is efficient and effective.

This contribution may be extended along two directions. First, it may be interesting to explore the "historical" case, in which the private information to be generalized is a "source trajectory" instead of a "source point". This applies, for example, to location-based services in which different requests can be linked to the same user. Second, we believe that a general purpose publicly available library for spatio-temporal cloaking may be designed along the lines of our approach. The library may include a set of predefined counting functions with different semantics, and support the extension with new ones. It may be designed to compute the generalization directly on a mobile device with connection to predefined services or on a "generalization (trusted) server".

A Proofs

A.1 Proof of Property 1

Proof. By Definition 5, for any spatio-temporal area A, $I(A) \leq A$. Hence, $I(\mathcal{G}(p)) \leq \mathcal{G}(p)$. Consequently, due to Definition 2, $\Omega(\mathcal{G}(p)) \geq \Omega(I(\mathcal{G}(p)))$. Since, by Definition 6, for any source point p such that $\mathcal{G}(p)$ is defined, $\Omega(I(\mathcal{G}(p))) \geq k$, then $\Omega(\mathcal{G}(p)) \geq \Omega(I(\mathcal{G}(p))) \geq k$. Hence, by Definition 3, $\mathcal{G}(p)$ is a *SafeBox*. □

A.2 Proof of Property 2

In this proof we use $|A|_T$ and $|A|_S$ to denote the temporal and spatial projections, respectively, of a spatio-temporal area A.

Proof. By Definition 2 we show that, for each pair of spatio-temporal areas A, A', if $A \subseteq A'$, then $\Omega(A) \leq \Omega(A')$. Since $A \subseteq A'$, then $|A|_T \subseteq |A'|_T$ and $|A|_S \subseteq |A'|_S$. Consequently, for each object o such that $\exists t \in |A|_T$ and $loc(o,t) = p \in |A|_S$, it also holds that $t \in |A'|_T$ and $p \in |A'|_S$. Consequently,

$$\{o \in O \text{ s.t. } \exists t \in |A|_T \text{ with } loc(o,t) \in |A|_S\} \subseteq \{o \in O \text{ s.t. } \exists t \in |A'|_T \text{ with } loc(o,t) \in |A'|_S\}$$

Hence, by Definition 9, $\Omega(A) \leq \Omega(A')$. □

A.3 Proof of Property 3

In this proof we use $|A|_T$ and $|A|_S$ to denote the temporal and spatial projections, respectively, of a spatio-temporal area A.

Proof. By Definition 2 we show that, for each pair of spatio-temporal areas A, A', if $A \subseteq A'$, then $\Omega(A) \leq \Omega(A')$. Since $A \subseteq A'$, then $|A|_T \subseteq |A'|_T$ and $|A|_S \subseteq |A'|_S$. Consequently, for each object o, if

$$\exists i \in I \text{ s. t. } \exists t \in (i \cap |A|_T) \text{ and } loc(o,t) = p \in |A|_S$$

then it also holds that $t \in (i \cap |A'|_T)$ and $p \in |A'|_S$.

Consequently, for each object $o \in O$, it holds that:

$$\{i \in I \text{ s. t. } \exists t \in (i \cap |A|_T) \text{ and } loc(o,t) = p \in |A|_S\} \subseteq$$
$$\{i \in I \text{ s. t. } \exists t \in (i \cap |A'|_T) \text{ and } loc(o,t) = p \in |A'|_S\}$$

Hence, by Definition 11, $\Omega(A) \leq \Omega(A')$. □

A.4 Proof of Theorem 20

Proof. We first prove that *TopDownBasic* computes a generalization function and then we show that it computes a safe generalization function.

We prove that *TopDownBasic* computes a generalization function by showing that, for any source point p, time influence parameter α, $\Omega()$ function and integer value k, if *TopDownBasic*$(p, \alpha, \Omega(), k)$ does not return **fail**, then $p \in$ *TopDownBasic*$(p, \alpha, \Omega(), k)$. We prove by showing that variable c (that is eventually returned by the algorithm) contains p at each iteration of the algorithm. In the first iteration variable c is set to $S \times T$, so $p \in c$. In any iteration such that $p \in c$, either c is returned or c is updated to a new value c' such that $p \in c'$. Indeed, if c is not returned, it is partitioned in c_1 and c_2. Since $p \in c$, then either $p \in c_1$ or $p \in c_2$ and the value of c is updated to c_1, if $p \in c_1$ or to c_2, if $p \in c_2$.

We now show that *TopDownBasic* is a safe generalization function according to Definition 6. Indeed we show that for any source point p, time influence parameter α, $\Omega()$ function and integer value k, if *TopDownBasic*$(p, \alpha, \Omega(), k)$ does not return **fail**, then, given $A =$ *TopDownBasic*$(p, \alpha, \Omega(), k)$, it holds that $\Omega(I(A)) \geq k$.

To show this result, we first show that, if *TopDownBasic*$(p, \alpha, \Omega(), k)$ does not return **fail**, then, given $A =$ *TopDownBasic*$(p, \alpha, \Omega(), k)$, it holds that $I(A) = A$. By Definition 5, we show that, for each point $p' \in A$, *TopDownBasic*$(p', \alpha, \Omega(), k) = A$. We prove this by showing that, at each iteration of the algorithm, variable c contains both p and p'. The result follows since the termination condition $(\Omega(c_1) \geq k \wedge \Omega(c_2) \geq k)$ does not depend on the source point. In the first iteration, $c = S \times T$, so $p, p' \in c$. In any iteration such that $p, p' \in c$, then either c is returned or c is updated to a new value c' such that $p, p' \in c'$. Indeed, if c is not returned, it is partitioned in c_1 and c_2 and, if $p \in c_1$, then also $p' \in c_1$ and analogous for c_2. So the new value of c' contains both p and p'.

Since $I(A) = A$, it is now easy to show that $\Omega(I(A)) \geq k$ by showing that $\Omega(A) \geq k$. Indeed, if the algorithm does not return **fail**, then it returns a ST-cell c such that $\Omega(c) \geq k$. At the first iteration, $c = S \times T$ and $\Omega(c) \geq k$ (otherwise the algorithm would return **fail**). When any children c' of c is such that $\Omega(c') < k$ the algorithm terminates returning c so the algorithm, when not returning **fail**, always returns a ST-cell c such that $\Omega(c) \geq k$. Hence, as proved, $\Omega(I(c)) = \Omega(c) \geq k$ and consequently *TopDownBasic* is a safe generalization function. □

A.5 Proof of Theorem 21

Proof. By definition of *TopDownBasic* and *TopDown*(Algorithms 1 and 2) it is easily seen that both algorithms return **fail** only if $\Omega(S \times T) < k$. Hence, given a source point p, a time influence parameter α, function $\Omega()$ and an integer value k, *TopDownBasic*$(p, \alpha, \Omega(), k)$=**fail** if and only if *TopDown*$(p, \alpha, \Omega(), k) =$ **fail**.

If *TopDown*$(p, \alpha, \Omega(), k)$ does not return **fail**, then it returns a ST-cell c'. We now show that (1) $p \in c'$, (2) for each ST-cell $\hat{c} \supset c'$, for each child $\hat{c_1}$ of $\hat{c}$ it holds that $\Omega(\hat{c_1}) \geq k$ and (3) if c' is not a leaf, for at least one child $\bar{c}$ of c' it holds that $\Omega(\bar{c}) < k$. Consequently, by definition of Algorithm 1 if follows that c' is the result of *TopDownBasic*$(p, \alpha, \Omega(), k)$.

(1) In the first iteration of the **while** loop of *TopDown*, $p \in c$ since $c = S \times T$. In the following iterations $p \in c$, because, when iteration continues, variable c is set to c_1 that is the child of c containing p. When iteration terminates the algorithm returns either c or its parent, hence the result contains p.

(2) We first prove that $\Omega(c') \geq k$. If the algorithm terminates at Line 4 or Line 12, then it returns variable c and it holds that $\Omega(c) \geq k$. Otherwise, if the algorithm returns at Line 5 of Line 14, it returns the parent p_c of current cell c. Since the child c_2 of p_c is such that $\Omega(c_2) \geq k$ (otherwise the Algorithm would have returned at the previous iteration), then, due to monotonic property, $\Omega(p_c) \geq k$. Hence, in all cases, $\Omega(c') \geq k$.

Now, consider a generic ST-cell $\hat{c} \supset c'$. $\hat{c}$ has two children: $\hat{c_1}$ that contains p and $\hat{c_2}$ that does not contain p. Since $\hat{c_1} \supseteq c'$, due to monotonic property, it holds that $\Omega(c_1) \geq k$ (we just proved that $\Omega(c') \geq k$). For what concerns $\hat{c_2}$, let's assume, by contradiction, that $\Omega(\hat{c_2}) < k$. By definition of *TopDown*, the algorithm would return $\hat{c}$ or its parent, which is in contrast with the fact that the algorithm actually returns $c' \subset \hat{c}$.

(3) A non leaf cell can be returned at Lines 5, 12 and 14. If the algorithm returns at Line 5, then the result c' has a leaf child $\bar{c}$ such that $\Omega(\bar{c}) < k$. The same holds for termination at Line 14. When the algorithm terminates at Line 12, then it returns the current ST-cell c, and its child c_2 is such that $\Omega(c_2) < k$. □

A.6 Proof of Theorem 22

Proof of Theorem 22 easily follows from the proof of Property 4 and is detailed in Section A.6.2.

A.6.1 Proof of Property 4

Proof. We show that, for each ST-cell c, if *BottomUpInversion*$(c) = \emptyset$, then $I(c) = \emptyset$ otherwise *BottomUpInversion*$(c) = I(c) =$ *Residuals*(c).

Let's consider the set $res(c)$ of points of c such that, for each point $p \in res(c)$, *BottomUp*$(p) \supseteq (p)$. Clearly, for each point $p \in (c \setminus res(c))$ it holds, by Definition 5, that $p \notin I(c)$.

If *BottomUpInversion*$(c) = \emptyset$, then, by definition of *BottomUp*, for each $p \in res(c)$, at the iteration of *BottomUp* in which variable $currentSTcell$ is set to the value c at Line 5, it holds that variable $count$ is set to $zero$ in the following line. Hence, none of these points is generalized to c. It follows that $I(c) = \emptyset$.

Now, consider the case *BottomUpInversion*$(c) \neq \emptyset$. By definition of *BottomUpInversion*, *BottomUpInversion* $\neq \emptyset$ only if $\Omega($*Residuals*$(c)) \geq k$. In this case *BottomUpInversion*$(c) =$ *Residuals*(c). By definition of *BottomUp*, for each point p in $res(c)$ then *BottomUp*$(p) = c$, since $\Omega($*BottomUpInversion*$(c)) \geq k$. Consequently, by Definition 5, $I(c) = res(c)$. We now

show that, for each ST-cell c, *Residuals*$(c) = res(c)$. We prove by induction on the height h of c.

Base case. If $h = 0$, then c is a leaf, hence, by definition of res, $res(c) = c$. By definition of *Residuals*, *Residuals*$(c) = c$.

Induction case. If $h > 0$ then we assume that for each ST-cell c' at height $h - 1$ it holds that $res(c') =$ *Residuals*(c'). In particular, let c_1, c_2 be the two children of c.

By definition of *BottomUpInversion*, if $\Omega($*Residuals*$(c_1)) \geq k$, then $\Omega($*BottomUpInversion*$(c_1))$ $\geq k$ and hence, for every point p in $res(c_1)$ it holds that *BottomUp*$(p) = c_1$ (by definition of *BottomUp*), consequently $p \notin res(c)$. Analogous for c_2.

Vice versa, if $\Omega($*Residuals*$(c_1)) < k$, then *BottomUpInversion*$(c) = \emptyset$ and hence for each point $p \in res(c_1)$ it holds that $p \in res(c)$. Analogous for c_2.

Overall:

$$res(c) = \begin{cases} res(c_1) \cup res(c_2) & \text{if } \Omega(res(c_1)) < k \text{ and } \Omega(res(c_2)) < k \\ res(c_1) & \text{if } \Omega(res(c_1)) < k \text{ and } \Omega(res(c_2)) \geq k \\ res(c_2) & \text{if } \Omega(res(c_1)) \geq k \text{ and } \Omega(res(c_2)) < k \\ \emptyset & \text{otherwise} \end{cases}$$

By definition of *Residuals*, *Residuals*$(c_1) \subseteq$ *Residuals*(c) if $\Omega($*Residuals*$(c_1)) \geq k$, otherwise *Residuals*$(c_1) \cap$ *Residuals*$(c) = \emptyset$. Analogous for c_2. Consequently *Residuals*(c) equals to:

$$\begin{cases} \textit{Residuals}(c_1) \cup \textit{Residuals}(c_2) & \text{if } \Omega(\textit{Residuals}(c_1)) < k \text{ and } \Omega(\textit{Residuals}(c_2)) < k \\ \textit{Residuals}(c_1) & \text{if } \Omega(\textit{Residuals}(c_1)) < k \text{ and } \Omega(\textit{Residuals}(c_2)) \geq k \\ \textit{Residuals}(c_2) & \text{if } \Omega(\textit{Residuals}(c_1)) \geq k \text{ and } \Omega(\textit{Residuals}(c_2)) < k \\ \emptyset & \text{otherwise} \end{cases}$$

The thesis follows since, by induction assumption, *Residuals*$(c_1) = res(c_1))$ and, analogously, *Residuals*$(c_2) = res(c_2))$.

□

A.6.2 Proof of Theorem 22

Proof. We first prove that *BottomUp*computes a generalization function and then we show that it computes a safe generalization function.

We prove that *BottomUp*computes a generalization function by showing that, for any source point p, time influence parameter α, $\Omega()$ function and integer value k then $p \in$ *BottomUp*$(p, \alpha, \Omega(), k)$, if *BottomUp*$(p, \alpha, \Omega(), k)$ does not return **fail**. In Line 1 of the algorithm, variable $currentSTcell$ is set to the leaf ST cell that contains p, hence, clearly $p \in currentSTcell$. In the **while** loop, variable $currentSTcell$ is updated with the parent of the previous value, hence at any iteration, $p \in currentSTcell$.

We now show that *BottomUp*is a safe generalization function according to Definition 6. Indeed we show that for any source point p, time influence parameter α, $\Omega()$ function and integer value k, then, given $A =$ *BottomUp*$(p, \alpha, \Omega(), k)$, it holds that $\Omega(I(A)) \geq k$, if *BottomUp*$(p, \alpha, \Omega(), k)$ does not return **fail**.

The thesis easily follows by the fact that, if *BottomUp* does not returns **fail**, then it returns a ST-cell c such that $\Omega(\mathit{BottomUpInversion}(c)) \geq k$. Since $\mathit{BottomUpInversion}(c) = I(c)$ (by Property 4), it holds that $\Omega(I(c)) \geq k$. □

References

[1] O. Abul, F. Bonchi, and M. Nanni. Never Walk Alone: Uncertainty for Anonymity in Moving Objects Databases. In Proc. of the 24th Int. Conf. on Data Engineering. IEEE, 2008.

[2] C. Ardagna, M. Cremonini, E. Damiani, S.De Capitani di Vimercati, and P. Samarati. Location Privacy Protection Through Obfuscation-based Techniques. In Data and Applications Security XXI, pages 47-60. Springer, 2007.

[3] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting Anonymous Location Queries in Mobile Environments with PrivacyGrid. In Proc. of the 17th Int. Conf. on World Wide Web, pages 237-246. ACM, 2008.

[4] L. Bertolaja, D. Ahmetovic, and S. Mascetti. Gonio, Aequus and Incognitus: Three Spatial Granularities for Privacy-Aware Systems. In Proc. of the 14th Int. Conf. on Mobile Data Management. IEEE, 2013.

[5] C. Bettini, S. Jajodia, X. S. Wang, and P. Samarati. Privacy in Location-Based Applications. Springer, 2009.

[6] M. L. Damiani, E. Bertino, and C. Silvestri. The PROBE Framework for the Personalized Cloaking of Private Locations. Transactions on Data Privacy, 3(2):123-148, 2010.

[7] A. Fattori, A. Reina, A. Gerino, and S. Mascetti. On the Privacy of Real-World Friend-Finder Services. In Proc. of the 14th Int. Conf. on Mobile Data Management. IEEE, 2013.

[8] B. Gedik and L. Liu. Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms. Mobile Computing, IEEE Transactions on, 7(1):1-18, 2008.

[9] G. Ghinita, M. L. Damiani, C. Silvestri, and E. Bertino. Preventing Velocity-based Linkage Attacks in Location-Aware Applications. In Proc. of the 17th Int. Conf. on Advances in Geographic Information Systems. ACM, 2009.

[10] 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. ACM, 2003.

[11] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing Location-Based Identity Inference in Anonymous Spatial Queries. Transactions on Knowledge and Data Engineering, 19(12):1719-1733, 2007.

[12] S. Mascetti, C. Bettini, D. Freni, and X. S. Wang. Spatial Generalization Algorithms for LBS Privacy Preservation. Journal of Location Based Services, 2008.

[13] S. Mascetti, C. Bettini, X. S. Wang, D. Freni, and S. Jajodia. ProvidentHider: An Algorithm to Preserve Historical k-Anonymity in LBS. In Proc of the 10th Int. Conf. on Mobile Data Management. IEEE, 2009.

[14] S. Mascetti, D. Freni, C. Bettini, X. Wang, and S. Jajodia. Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. The VLDB Journal, 20(4):541-566, 2011.

[15] 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 Int. Conf. on Very large data bases. VLDB Endowment, 2006.

[16] L. Šikšnys, J. R. Thomsen, S. Šaltenis, and M. L. Yiu. Private and Flexible Proximity Detection in Mobile Social Networks. In Proc. of the 11th Int. Conf. on Mobile Data Management. IEEE, 2010.

[17] M. Wernke, P. Skvortsov, F. Dürr, and K. Rothermel. A classification of location privacy attacks and approaches. Personal and Ubiquitous Computing, pages 1-13, 2012.

[18] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three protocols for location privacy. In Privacy Enhancing Technologies, pages 62-76. Springer, 2007.