UNIVERSITÀ DEGLI STUDI DI MILANO

Ph.D. School in
Computer Science

Computer Science Department

Ph.D. in
Computer Science
XXVII° Cycle

Location sharing: privacy threats and protection

INF/01

Ph.D. candidate:
Letizia BERTOLAJA

Advisor:
Prof. Claudio BETTINI
Co-Advisor:
Dr. Sergio MASCETTI

School Director:
Prof. Ernesto DAMIANI

Academic Year 2013/14

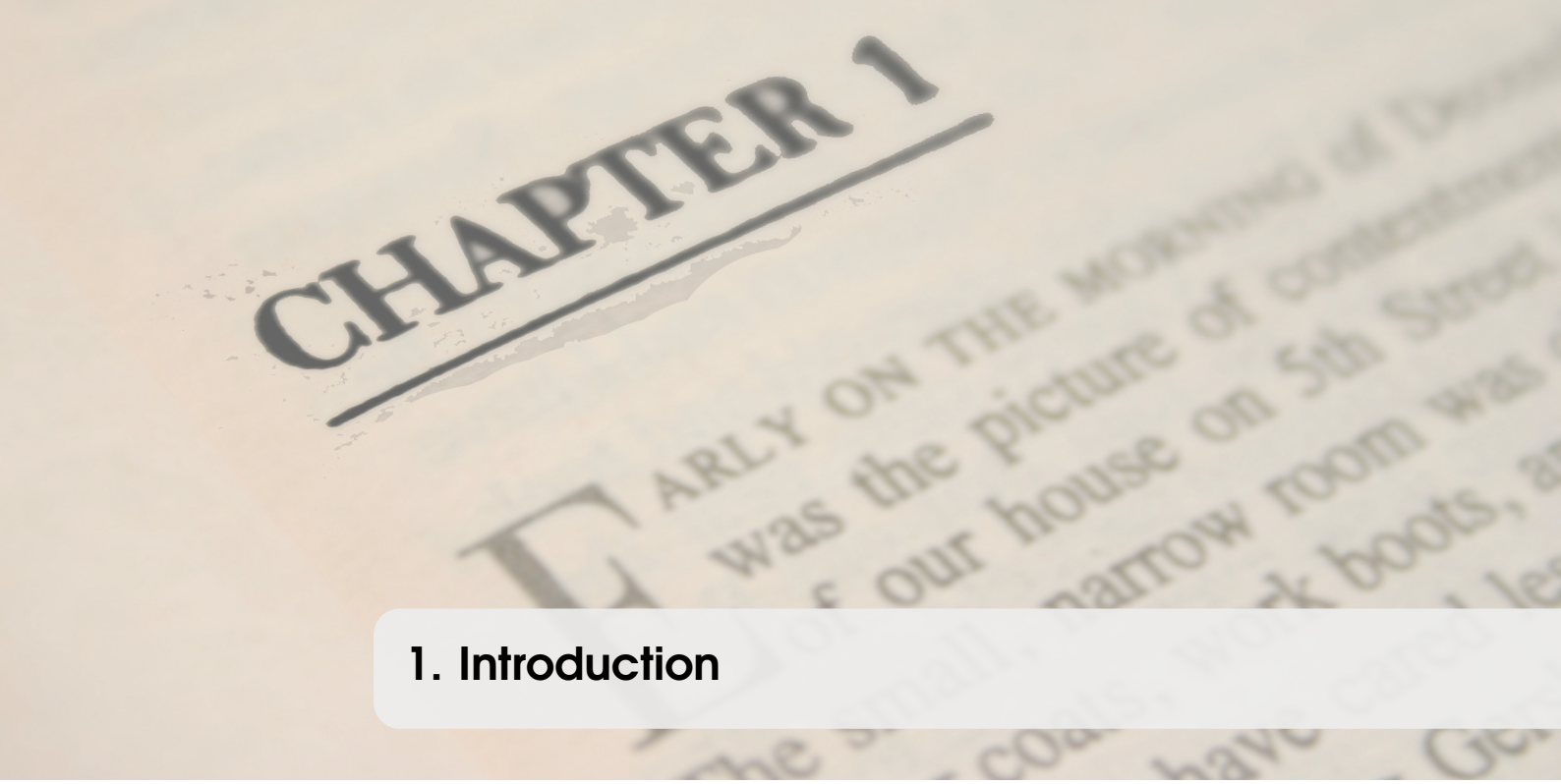
Location sharing: privacy threats and protection

Letizia Bertolaja

Advisor: Prof. Claudio Bettini

Co-Advisor: Dr. Sergio Mascetti

Contents



1. Introduction

In recent years there has been a growing increase in the number of users that use smartphones, tablets, wearable technologies and other devices that users have with them constantly [O1][O4]. The numbers and some statistics talk themselves: in 2013 Apple's App store reached the number of 1 million apps and, according to the AppBrain Stats the current[1] number for Google Play's applications is about $1,3$ million. Each of this application tries to fulfil a particular need for users throughout the world. Exploiting built-in sensors, mobile devices offer to their users almost every service for every need.

In particular, the capability of these latest generation mobile devices to detect the position of the users has led to the emergence of ad-hoc services as well as geo-aware social networks (GeoSN). Hundreds of location-based applications are now available: let's think about range queries, like where is the nearest ATM or gas station? If we are in a unknown city this kind of services can be really precious and time-savings. User's location can also be exploited to give targeted advertisements of shops, restaurants, etc. It is useful for the business activity to contact close users, since they have more chance to go, for instance, to their restaurant; likewise, a user will be probably more interested in knowing the special offer of a restaurant within $200m$ than one at $20km$ from his position. Even the "social life" of a user can be enhanced by the use of location information: for instance the friend finder applications allow to know which of the user's friends are in his proximity. The close friends can eventually communicate and even decide to meet, since they are near. Furthermore, in the recent years, most of the major social networks, like Facebook, Twitter, Google+, Foursquare, offer to their users the possibility of geo-tagging the published contents, giving to the users new ways to exploit and share their positions. For example a user can post a new status together with the user's geographic coordinates, or publish a photo with the position of where it has been taken.

Reading about all the benefits in sharing location, it might suggest that sharing it more and

[1] data retrieved at the beginning of September 2014

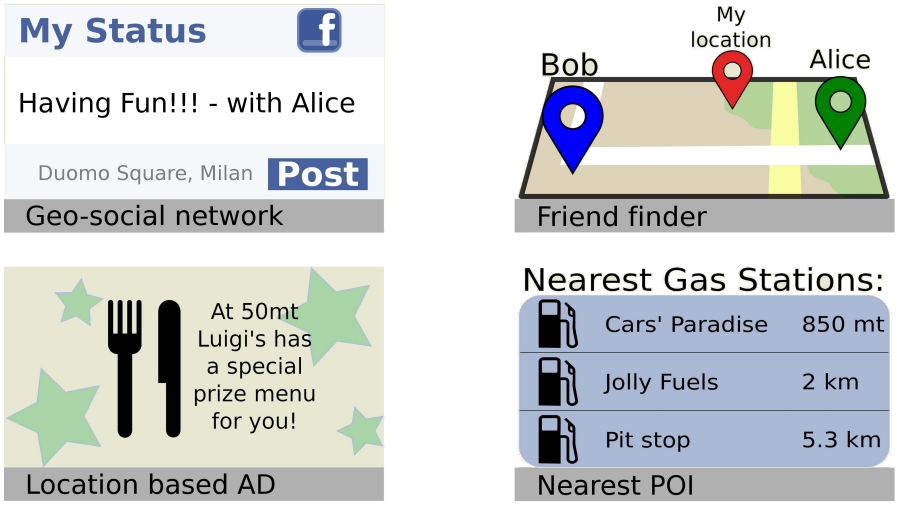


Figure 1.1: Common Location Based Services

more is always recommendable and advantageous. Unfortunately, there are several practical cases that unveil the danger of sharing location indiscriminately. For instance, let's suppose that a user has just told everyone that he is on vacation (and not at his house): if he adds how long his trip is, then thieves know exactly how much time they have to rob him. Many contributions in the scientific literature have shown how through the location information it is possible to infer several information about the user. It has been shown that it is possible to identify user's identity, if he is anonymous in the LBS, and, if the user is not anonymous, it is feasible to infer user's home location, habits and also politic preferences and sexual orientation [23] [13]. User's work and home location information may be used to stalk him and even rob him, as shown by Borsboom et al. through the provocative website "pleaserobme.com". The problem is evident when sensible information about the user are involved. For example, if a user is frequently geo-tagged in the headquarters of a political party it is trivial understanding which party he is going to vote for. If this information is public it can be used against the user: a company may decide not to hire him due to his political preferences. A similar reasoning does apply for sexual orientation (that again can be inferred through user's locations): whatever the context, no one wants to be discriminated based on this information, that is strictly *private* and it's up to the individual to decide, if and when, sharing it.

Indeed private, according to the Oxford English Dictionary[2], means "restricted to one person or a few persons as opposed to the wider community; largely in opposition to public". Keeping some information and data about ourselves private is a right and indeed privacy is one of the fundamental rights in the Universal Declaration of Human Rights [O7]. There is debate about whether privacy is or not a widespread concern for, in general, the individuals and, specifically, for the users. Settling the debate is out of the scope of this dissertation, but it is undeniable how privacy issues are nowadays more frequent, thanks to the introduction of new habits and

[2]http://www.oed.com/

technologies, as discussed before. Some users are now so concerned about their privacy to decide quitting social networks that do not protect their personal data [31]. Indeed the simplest solution for privacy-concerned users is to stop sharing data, i.e. quit the social network or stop using the LBS, but in most of the cases this is not what the user, and the social network or LBS, want.

The scientific literature reflects this concerns, proposing many contributions that deal with privacy, in general, and location privacy, specifically. The location privacy issues in using LBS or Geo-Social networks are not a new topic, since these problems have been extensively studied in the previous years, and many works both formalise location privacy issues and propose privacy protection techniques for many applications, as we will see thoroughly in Chapter 2. The first approach in the literature to protect location privacy, i.e. anonymity, protects the "identity privacy", in which the anonymity of the user must be preserved, such that the location information is not used as a quasi-identifier [25][18][29]. In the anonymity approach, the main technique makes use of the principle of k-anonymity: a user is considered "k-anonymous" if his position is indistinguishable from the locations of at least $k-1$ other subjects [25]. In this approach anonymity is achieved by sending to the SP, instead of the user's punctual location, a region which is sufficiently large to include, in addition to the issuer, other k-1 potential users, as first proposed in 2003 by Gruteser [25]. The second family of techniques (i.e. obfuscation techniques) protects the "location privacy": in this approach the user's location is considered sensitive and thus the data to be protected [61][43]. These techniques assume that the adversary, i.e. the one whom the users do not trust, may know the identity of the user but not his location, or at least not precisely.

Given this wide interest in location privacy, what is the motivation and the originality of a new contribution on location privacy as the present dissertation? First of all, we are far for a comprehensive solution that takes into account all the location privacy issues in every LBS or Geo-Social networks. Many privacy preserving techniques protect user's privacy only under specific assumptions (e.g. uniform distribution of the users in the space) that are not always realistic. Also, new privacy breach are emerging: for instance, the birth of the Geo-social networks have created new privacy threats (that we will discuss later in Chapter 2) that did not exist in LBS: only some of these threats are already formalised and the corresponding privacy protection techniques are not always been provided. Moreover, the location privacy scenario is complicated by the fact that today a user is likely to use more than one social network: this dispersion of user's information over multiple SNs makes the control over his shared data more difficult. The decentralisation of data makes the user unaware of the volume and the precision of his shared data. Questions like: "Which information am I sharing?", "With whom?", "What can other users learn by the aggregation of data in different SNs?" are not trivial in the current context. Furthermore, the attention in location privacy can not be considered purely academic or scientific: privacy topics had always been discussed in primis in the legal field. The privacy protection has a long forensics history, rooted in the ancient Greek philosophy, when a distinction was made between the 'outer' and the 'inner', between public and private, between society and solitude [27]. Most countries protect privacy by laws and rules: see for example The Privacy Act of 1974 of the USA [O6], or European Data Protection Directive [O2] and the more recent General Data Protection Regulation proposal [O3]. In the past connections between the legal and the scientific areas were limited, and these two worlds, even if speaking about common themes,

as privacy, were not communicating. The dialogue between these worlds is now lively [45][48], since the legal framework needs aid from the "technician" to help protecting privacy and the scientific community can not protect users' privacy if there is an indiscriminate use of individual's private data. The discussion on the proposal of General Data Protection Regulation [O3] has involved many anonymity techniques (also specific for location privacy), that were considered and analysed. This is an excellent example that 1) shows how privacy preserving techniques are not a mere academic exercise but are taken into account in the making of laws and, therefore, 2) it adds motivations in nowadays working in privacy, as computer scientists.

1.1 Summary of results

This dissertation deals with location privacy in Location Based Services and Geo-Social Networks. Results from this dissertation appear in the following publications: [5, 37, 38, 39]. These are the main contributions:

1. Motivate the importance of the location privacy topic by identifying the privacy threats of sharing locations

   T-I. Distance preserving transformations can be used as privacy preserving techniques in proximity services [40], under the assumption that the users are uniformly distributed in the space. In the context of privacy preserving data mining, it has been shown that distance preserving transformations can be subject to privacy attacks in presence of prior knowledge of the adversary in terms of input-output pairs and samples from original or similar dataset [34]. In Section 3.1 we show through a location privacy attack which are the practical conditions that make distance preserving transformations unsafe, in a realistic environment.

   T-II. As discussed before, new location privacy threats emerged with the spread of Geo-Social networks. Through these social networks it is possible to observe multiple users' presence in the same place. Since a user can consider a privacy threat to be located with someone with whom he does not want to, in Section 3.2 we present, classify and formalise the co-location privacy threat.

2. Propose privacy preserving techniques and tools:

   P-I. In contribution T-I we show the weakness of distance preserving transformations used as privacy preserving techniques, even in a practical scenario (i.e. users non-uniformly distributed). In Chapter 4 we propose a novel privacy preserving technique, that, thanks to a space transformation, can make distance preserving transformation safe, even in a space in which the users' distribution is not uniform.

   P-II. Obfuscation techniques, among others in the literature (see Chapter 2), can be used to protect user's privacy in many location based services. In the family of the obfuscation techniques, many privacy preserving approaches perform a spatial generalisation of the precise user's position in order to decrease the sensitivity of the location information. A common problem is that spatial generalisations are used, often implicitly, as a tool, without extensively describing their properties

and basic operations. In Section 5.2 we propose two spatial generalisation, Gonio and Aequus, specifically designed for location privacy techniques. Gonio is a naive subdivision in which each cell shares the same angular dimension, with the result that cells at different latitudes covers areas of different size. The proposed Aequus cells have the same area (hence granting the same level of obfuscation), independently from their latitude. For both Gonio and Aequus we study the main properties and we show how to efficiently compute the basic operations.

P-III. In Section 5.3 we propose another generalisation, called SafeBox. A SafeBox is a spatio-temporal generalisation of the user's location in a given time, with the property that in the generalised area there are at least k objects. These "objects" can be users, obtaining in this way a k-anonimity technique, or number of shops, categories of building, etc.. We provide a general notion of object counting function supporting different semantics. We design a bottom-up algorithm, that to the best of our knowledge is the first safe generalisation algorithm adopting a bottom-up strategy. We'll show through the experiments how a bottom-up strategy can be advantageous with respect to a top-down one, especially if the objects to count are retrieved through online queries.

1.2 Outline

Related work in location privacy in LBS and Geo-Social networks is discussed in Chapter 2. The contributions of this dissertation in highlighting location privacy threats are presented in Chapter 3, contribution T-I in Section 3.1 and contribution T-II in Section 3.2. The privacy protection topics are reported in two chapters: the privacy protection technique obtained through a space transformation (contribution P-I) is detailed in Chapter 4, while the techniques that generalise users' locations (contributions P-II and P-III) are presented in Chapter 5. In Chapter 6 we conclude the dissertation discussing the open problems and summarising the contributions of this work.

2. Related work

The goal of this chapter is to present the state of the art in location privacy protection. As mentioned before, there are dozens of contributions in the literature that deserve to be presented in a comprehensive survey, as [58] and [6]. However, in this chapter we give an overview of the privacy protection proposals, presenting only some contributions, without the presumption of being exhaustive.

In Section 2.1 it is presented the state of the art on privacy in Location Based Services (LBS), while in Section 2.2 are discussed some specific contributions for privacy in Geo-social networks.

2.1 Location privacy in LBS

To better present the location privacy literature in LBS, we first should identify which is the subject of the protection and the entity that is threatening it. The *protection subject* is the data to protect. As in [44], we distinguish between protection of the user's identity and protection of the location shared by the user. As we will see in this chapter, protecting identity rather than the location leads to very different techniques.

Once it is identified the protection subject, it is important to bear in mind that each privacy protection proposal assumes, implicitly or explicitly, an *adversary* from which protecting the users. The adversary is any entity that can potentially have access to user's data and whose intention is to infer his location, private data or identity, threatening his privacy. It can be, for example, a provider of a Location Based Service (LBS), an eavesdropper that intercepts the communications 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. Wider it is the knowledge that the adversary can use to violate user's privacy, more threatening can be his inferences on user's data and, consequently, the proposal of the corresponding privacy protection can be more challenging.

2.1.1 Protecting identity information

The first approach, i.e. anonymity, protects the "identity privacy", in which it is the anonymity of the user that must be preserved. Indeed, the location information can be used as a quasi-identifier, i.e. together with other external data, compromising user's identity (that is assumed as anonymous in the LBS) [25][18][29]. The user's identity can then be associated with sensitive information related with the LBS, damaging his privacy.

In the anonymity approach, the main technique makes use of the principle of k-anonymity: a user is considered "k-anonymous" if his position is indistinguishable from the locations of at least $k-1$ other subjects, as first proposed in 2003 by Gruteser [25]. In this approach anonymity is achieved by sending to the SP, instead of the user's punctual location, a region that is sufficiently large to include, in addition to the issuer, other k-1 potential users [29][18]. In this way the probability of identifying the user's identity is $\frac{1}{k}$. Chow et al. [8] propose a technique in which the user can specify a number k of users (to be anonymous with), and the minimum A_{min} and maximum A_{max} area of the cloaking region. In this way, even if the user is located in a area with high users density the cloaked region will not be too small, since it will be at least large as A_{min}. In [41], Mascetti et al. propose a spatial cloaking technique: through a top-down approach, each region returned by this technique contains at least k object. In [43] it is introduced the concept of *historical* k-anonymity: in this technique also the temporal component is considered for computing the k-anonymous area, since the technique is designed for moving objects.

Some variants of k-anonymity technique have been proposed later. The l-diversity idea [36] stems from the limits of k-anonymity principle. Let's suppose that the location of the user u is enlarged to an area including other $k-1$ users, but, all of them are located within a hospital. Then, an adversary, knowing that u is in the set of that k users, can infer that u is located in the hospital and that he is ill. To overcome this problem, Bamba et al [3] propose a cloaking algorithm in which the location of the user is k-anonymous and further, it is unidentifiable from a set of l static locations or symbolic addresses. Similarly, the t-closeness [33] and the p-sensitivity [55] concepts are further extensions of the l-diversity idea.

2.1.2 Protecting location information

The second family of techniques (i.e. obfuscation techniques) protects the "location privacy": in this approach it is the user's location that is considered sensitive and thus the data to be protected [61][43]. These techniques assume that the adversary may know or infer the identity of the user but not his location, or at least not precisely. The user's location is not shared with precision but it is "modified": by using the LBS, the user's location is often enlarged (or fake locations are used) in order to decrease its sensitivity. The knowledge of this region should not provide any additional information on the user's actual position within.

In [30] users generate several fake positions data and send them to the SP with the true position data of users. Since the SP could not distinguish between fake or real locations, the location privacy is preserved. This approach has the drawback of having a high communication and computational costs since a number of service requests are fake.

In [52] and [40] the computation of the proximity between users is performed using distance-preserving transformations. With distance preserving transformations, the locations of users A and B are modified, but the real distance $d(A,B)$ is preserved: sharing the modified locations,

even with an untrusted entity (e.g. the service provider), does not threaten the users' privacy, since their real locations are undisclosed. Unfortunately, this holds only when it is assumed that the adversary does not know the distribution of the users in the space. Indeed in Section 3.1 we will show how an adversary can learn the approximate position of most of the users, that were protecting their privacy using such distance preserving techniques.

Another approach in space obfuscation techniques is to generalise user's location, by enlarging it into a region, in order to avoid sharing his precise position, that is the information to protect. In [2] the position of the user it is enlarged to a circular area and then shifted before sending it to the service provider, to avoid reverse engineering attacks. FriendLocator [56] technique assumes the existence of a subdivision of the space into regions or "cells". Two users are considered in proximity if they are located in the same cell or, at most, two adjacent cells in the grid, where the "size" of the cells are computed considering the proximity threshold shared by the users. This techniques computes the proximity incrementally, and this may lead to a violation of user's privacy. Among other works whose contribution is aimed at the protection of privacy in the calculation of proximity, Zhong et al. [62] propose three different techniques called Louis, Lester and Pierre. These are centralised protocols in which security is guaranteed by the use of public key cryptography. While in Louis and Lester protocols the proximity is computed through a trusted third party (Louis) or by one of the users (Lester), the Pierre protocol is based on a spatial subdivision of the space into a grid, with each cell having edge equal to the requested distance threshold. The users are then considered in proximity only if they are located in the same cell or in two adjacent cells. The protocol *Louis* reveals to the server the proximity information, assuming that this information may not lead to a location privacy breach. This assumption should be probably reconsidered in light of the results that will be presented in Section 3.1.

Many works rely on a spatial subdivision as a basis upon which computing the spatial generalisation of the user's position, as we will discuss it thoroughly in Chapter 5. In spatial generalisation techniques there is an evident trade-off between privacy and utility. Let's assume that the users is performing a query to the untrusted service provider (SP) in order to get the nearest ATM: if he shares with the SP a generalised area big as the whole city, then his privacy is for sure safe, but the query results are not so useful; conversely if the user gives to the SP an area of $10m^2$, the nearest ATM is precisely found but the user's privacy can be threatened.

In [57] a multi-level partitions of the spatial domain is performed in order to obtain a secure proximity detection through a client-server solution. Precisely, it is performed a subdivision of the spatial domain into a number of non-overlapping regions, called granules. In this proposal there are several list of granularities that are organised into levels, similarly as we will propose in Section 5.2 with the Gonio granularity family. In addition, in that Section we will describe also the basic operation between granules.

In [25] both the temporal and the spatial dimensions are taken into account when generalising the user's location. In this work the user's location generalisation is performed in order to obtain a k-anonymous area: the spatial resolution is improved by some seconds reduction in the temporal resolution. Also Bamba et al. [3] propose a spatio-temporal cloaking for supporting anonymous location-based queries. In this proposal each user can explicitly define his preferred location privacy requirements in terms of both location hiding measures (e.g., location k-anonymity and location l-diversity) and location service quality measures (e.g., maximum spatial resolution

and maximum temporal resolution). As we will discuss in Chapter 5, the technique proposed in this paper protects the user's privacy only under the assumption the adversary does not know the protection technique and the users' distribution in the spatial domain: through the SafeBox proposal (see Section 5.3) we will overcome this privacy threat.

The temporal component becomes crucial, when we deal with user's trajectory instead of punctual positions. Ghinita et al. [20] present a spatio-temporal cloaking in which the user is protected against linkage attacks. In these attacks the exact locations of the users can be inferred based on prior knowledge about maximum user velocity and roads/traffic constraints. Instead, to avoid privacy breach in case of low density areas, the user's trajectory in [26] is modified through an *uncertainty-aware path cloaking*, capable of hiding some location samples of the trajectory. In another scenario, if users' trajectories are published in a database then the location information is sufficient to identify the users identity [54]. To avoid this threat, the authors in [54] present a greedy anonymization algorithm that generalise the trajectories (suppressing certain points): the anonymization is performed such that the utility of the dataset as accurate as possible, while guaranteeing users' privacy.

Approaches that differs completely from the ones presented so far, are the cryptography based ones. In [44] the authors propose two new protocols to compute users' proximity. The first protocol, called C-Hide&Seek, provides complete protection with respect to the SP and satisfies the privacy requirements of each user with respect to her buddies. The second protocol, called C-Hide&Hash, offers the same guarantees, but provides an even higher level of privacy with respect to the buddies at the cost of higher communication and computation costs. In [21] the users' privacy performing location based queries is based on Private Information Retrieval (PIR). The safeness of the used *computational* PIR, which employs cryptographic techniques, relies on the fact that it is computationally intractable for an attacker to decrypt the query sent to the SP by the user. A more recent solution [32] exploits encryption and multi-level spatial cloaking to grant privacy-preserving proximity detection. In this approach it is possible to dynamically trade accuracy for communication cost in a quantitative manner.

2.2 Location privacy in Geo-Social Networks

The study of concerns raised by the use of Geo-Social Networks is relatively newer, since the spread of this kind of social networks is more recent, compared with "traditional" LBS.

The privacy implications of geo-tagging have been presented in [15]: they showed how easy it is to correlate geo-tagged data with corresponding publicly available information, uncovering the ease in violating users' privacy. Analogously, Pontes et al. [50] perform a study showing the feasibility of inferring users' home locations trough the data of three of the most popular geo-social networks: Foursquare, Google+ and Twitter.

In [35], the authors present a privacy management system named *MobiShare* that allows location sharing between both trusted social relations and untrusted strangers, where the privacy protection is given by a location encryption. In [51] the authors identify four location privacy aspects peculiar for Geo-Social Networks - location, absence, co-location, and identity privacy - and discuss about possible ways to protecting privacy when such threats occur. Freni et al. [14] highlight the privacy concerns raised by the use of geo-tagging in social networks, in particular

they focus on location privacy and absence privacy threats. The paper formalizes the setting, provides a foundation for easily defining privacy preferences, and provides techniques that generalize the tags of resources so that these remain useful while ensuring that the privacy preferences are enforced. These techniques exploit spatial and temporal generalization, and they utilize publication delays.

The privacy threat that, to the best of our knowledge, is not formalised properly in the literature is the co-location privacy threat: in Section 3.2 we will formalised it and analyse the circumstances under which it can stem in a social network scenario.

3. Privacy threats

In this chapter we will consider two different types of privacy threats. Having a better knowledge of privacy threats can help us to better understand the motivation for studying privacy issues, constructing solid bases on which building protection techniques.

In Section 3.1 we first analyse the practical conditions that can threaten users' privacy in using proximity services, whose privacy protection is provided through distance preserving transformations. This section is based on the following publications: [38][37]. Instead, in Section 3.2 we study a social network-related threat, the co-location privacy threat, stemmed from the user's habits of sharing their locations with the social network's friends.

3.1 A practical location privacy attack in proximity services

The aim of *proximity services* is to raise alerts based on the distance between moving objects. While distance can be easily computed from the objects' geographical locations, privacy concerns in revealing these locations exist, especially when proximity among users is being computed. *Distance preserving transformations* have been proposed to solve this problem by enabling the service provider to acquire pairwise distances while not acquiring the actual objects positions. It is known that distance preserving transformations do not provide formal privacy guarantees in presence of certain background knowledge but it is still unclear which are the practical conditions that make distance preserving transformations "vulnerable".

We study these conditions by designing and testing an attack based on public density information and on partial knowledge of distances between users. A clustering-based technique first discovers the approximate position of users located in the largest cities. Then a technique based on trilateration reduces this approximation and discovers the approximate position of the other users. Our experimental results show that partial distance information, like the one exchanged in a friend-finder service, can be sufficient to locate up to 60% of the users in an area smaller than a city.

3.1.1 Introduction & motivation

Proximity services aim at notifying a user when another user or, more generally, an object of interest to the user happens to be spatially closer than a certain threshold distance. Examples of these services are so called *friend-finder* services. They can be efficiently implemented by a server receiving position information from each moving object and sending an alert when the distance threshold condition is verified.

When users are the considered moving objects and the server is untrusted, the acquisition of exact location information by the server is a privacy threat. The problem of privacy preservation in location based services has been extensively studied in the literature [6, 19], but mostly focusing on services identifying the location of points of interest. Since a proximity service can be provisioned simply based on the distance between the two moving objects, an intuitive way to preserve location privacy is to use fake locations for the objects, with coordinates appropriately chosen in order to retain the correct value or a good approximation for the relative distance. We consider the server provisioning the service as a semi-honest entity, in the sense that it follows the given protocol but it may attempt to infer the locations of the users. A natural question arises: is distance information sufficient for an adversary to find the actual location of a user? Technically, this is equivalent to ask if distance preserving transformations are unsafe. If users are uniformly distributed in the geographical space and no other background knowledge is available, the transformations are actually safe, since the same set of distances can be observed for many different assignments of specific positions to the users. This uniform distribution assumption is implicitly or explicitly made in several papers, as described in the following.

In [62] three secure computation protocols are proposed to notify a user about the proximity of another user. Communications among the user devices occurs through a server; One of the protocols, named *Louis* reveals to the server the proximity information, assuming that this information may not lead to a location privacy breach. This assumption should be probably reconsidered in light of our results. In [56], [57], [32] location privacy is achieved through spatial cloaking and encryption techniques, though these solutions reveal to the server approximate distance information. The *Longitude* protocol [40] allows users a finer control of location privacy with respect to other friends by specifying privacy preferences in terms of spatial granularities. *Longitude* also guarantees complete privacy with respect to the server under the assumption that it has no a priori knowledge on the distribution of users, i.e., when a uniform distribution is assumed. This assumption is indeed related to the fact that the server acquires approximate distance information. We find the assumption that distance information does not reveal the user actual position also in some work on general LBSs [28, 44]. In particular, in [28], spatial cloaking is used to guarantee k-anonymity that is computed using proximity information rather than the precise user locations. The paper proposes both a centralised algorithm and a distributed one. The centralised algorithm assumes a server acquiring proximity information for all users. Revealing proximity instead of exact positions is based on the intuition that information on distance is not sufficient to identify the position. Our findings would suggest that the centralised algorithm may be subject to privacy breaches in case of a semi-honest server.

In the context of privacy preserving data mining, it has been shown that distance preserving transformations can be subject to privacy attacks in presence of prior knowledge of the adversary in terms of input-output pairs and samples from original or similar dataset [34]. In our attack

the only background knowledge is the publicly available density distribution. The unsafety of distance preserving transformations in presence of certain kinds of background knowledge is also discussed in the context of secure kNN queries [59]. The authors consider as safe the case in which the adversary observes only the encrypted DB and the distances (called "level 1" attack). Our study shows that even this attack model can lead to privacy breaches when the distribution of the original data is known, as it is the case for population distribution.

Finally, in the context of private spatial join computation, a distance based attack in presence of background knowledge about spatial distribution shows that the position of isolated points (outliers) may be discovered [22]. A defence is proposed based on a spatial transformation that eliminates distances longer than a certain threshold in the transformed space. In principle, friend-finder services may be effective even if limited to short distances (2-3 km); however, our study shows (see Section 3.1.5) that even ignoring any distance larger than that threshold the attack we propose can still pose privacy threats. In addition, our study does not want to rule out services in which long distances are considered, and identifies under which conditions the adversary actually derives users location.

This section formalises a location privacy attack that, based on partial information on the distances between users and public knowledge on the average density of population, aims at approximately localising the users, independently on their fake position assigned by a privacy preserving algorithm. We achieve this goal by applying a distance-based clustering algorithm, and then matching obtained clusters with geographical regions with similar densities. Population density information drives the mapping process. We have reported preliminary results based on this idea in [37]. In this section, in addition to formalising the attack, we significantly enhance its effectiveness by introducing a trilateration technique, which refines the position of other users once the position of a few is determined. We show through an extensive experimental evaluation, considering geographical regions with substantially different density distributions, that the attack is effective in practice and that this technique, not only allows to position many more users, but it also position them with much higher precision.

We are presenting this attack assuming a worst case scenario for the users' privacy: indeed the adversary we simulate in this attack has a deep knowledge of the privacy protection technique, that he exploits together with the population distribution and partial information on the distances between users. The worst case scenario it is not always likely, but it helps us to better understand the limits of privacy protection techniques, that, in general, can not protect the users from any possible threat or adversary.

To summarise, our contributions are as follows:

1. Our study systematically explores the problem of identifying the position of moving objects based only on partial information on their distance and on background knowledge on the density distribution.

2. We provide a formalisation of the problem in terms of a privacy attack in presence of background knowledge

3. Our experiments on real geographic data and publicly available population density data show that the attack is effective even with partial distance information as typically acquired by running proximity services.

The rest of the Section is organised as follows. In 3.1.2 we formally describe the location re-identification attack. In 3.1.3 we describe the specific strategy used to make the attack practically feasible and we report the basic algorithms and several optimisations. In 3.1.5 we report our experimental results, and we conclude in 3.1.6.

3.1.2 Problem Modeling

We consider a service provisioned to a set of users by exchanging data through a service provider that only acquires information about the distance between some of the users.

Basic definitions

Let U be the set of users. Given a user $u \in U$, his position is denoted by $pos(u)$ that is a point in a discrete bi-dimensional space S. We denote with $d_p(p_1, p_2)$ the distance[1] between two points $p_1, p_2 \in S$.

Given the average density of population in each area at the precision available from public sources, we consider as *background knowledge BK* the information about how many users in U are in each area. More formally, the adversary can compute the function $nu : G \to \mathbb{R}$ returning the number of expected users contained in g for each cell g of a grid[2] G partitioning the spatial domain S. Given $nu()$, the probability that a generic user is located in a cell g is given by the number of users in g divided by the total number of users $|U|$.

Definition 3.1.1 The probability $P[u,g]$ that a user u is located in a cell g of G, based on BK only, is given by: $\frac{nu(g)}{|U|}$

Note that, since the adversary has no other background knowledge, $P[u,g]$ for a given g is the same for all users.

During the provisioning of the service, the adversary collects data about the distances between users. We call this information the *observation knowledge OK*, and we model it by the function $d_u : U \times U \to \mathbb{R}$ returning for a pair of users u_i and u_j the distance between their positions, i.e., $d_u(u_i, u_j) = d_p(pos(u_i), pos(u_j))$. Note that d_u is a partial function and it is undefined for those pairs of users whose distance in unknown to the adversary.

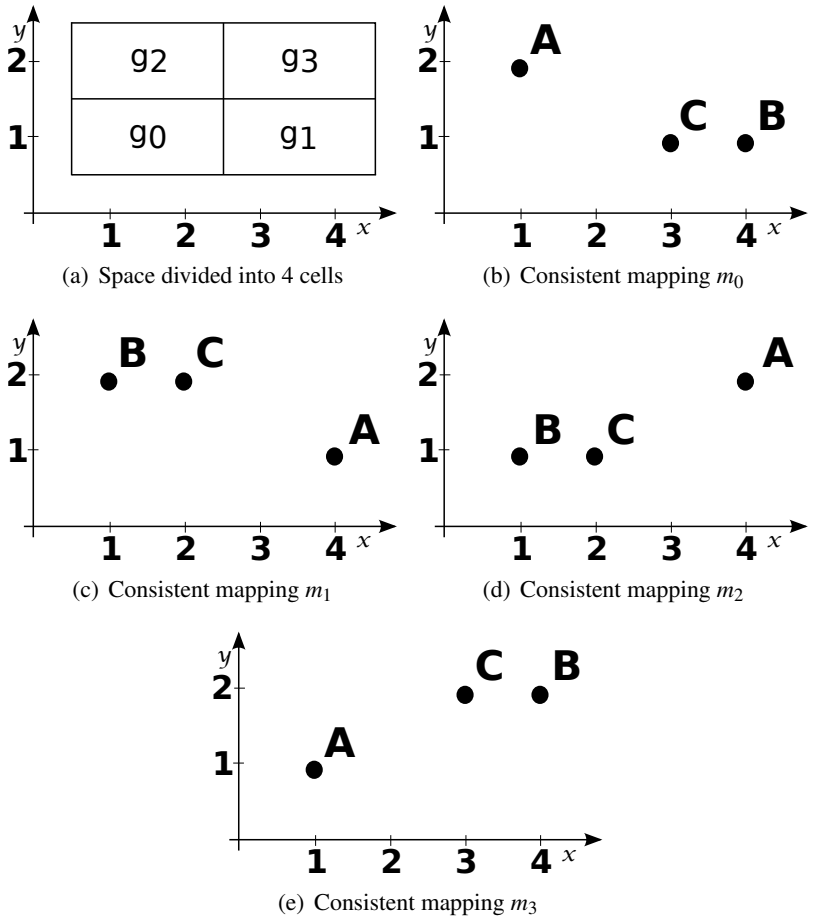
■ **Example 3.1** Consider a friend-finder service provided in a space S composed by a regular grid of 8 possible user positions (see Figure 3.1(a)). Four cells are defined in S and the probability that a generic user u is located in each cell is: $P[u,g_0] = P[u,g_1] = P[u,g_2] = 1/6$ and $P[u,g_3] = 1/2$. We consider 3 users: Alice (A), Bob (B) and Carl (C). Since Alice is friend of Bob and Carl, while Bob and Carl are not friends, two distances are known to the adversary: $d_u(A,B) = \sqrt{10}$ and $d_u(A,C) = \sqrt{5}$.

■

We now show how the adversary can exploit *BK* and *OK* to infer the position of the users.

[1] The distance $d_p()$ can be either the euclidean distance or the geographical distance computed on the geoid.

[2] Here we use a grid, but the more general concept of spatial granularity can be used as well.



(a) Space divided into 4 cells

(b) Consistent mapping m_0

(c) Consistent mapping m_1

(d) Consistent mapping m_2

(e) Consistent mapping m_3

Figure 3.1: Examples

Modelling an attack to discover user positions

We first define the concept of *user-to-point mapping m* to capture the intuition of the assignment of a specific geographical position to each of the users. The goal of the adversary is to identify the mappings representing the real position of as much users as possible. Formally $m : U \rightarrow S$ is a total function that associates each user with a geographical position.

By using OK, it is possible to restrict the set of *consistent* user-to-point mappings to those that do not violate any distance constraint defined in OK. We denote the set of all consistent user-to-point mappings with M.

Definition 3.1.2 A user-to-point mapping m is a *consistent user-to-point mapping* if, for each $u_i, u_j \in U$ such that $d_u(u_i, u_j)$ is defined, $d_u(u_i, u_j) = d_p(m(u_i), m(u_j))$.

■ **Example 3.2** Let's continue with Example 3.1 and let's consider a user-to-point mapping m such that: $m(A) = \langle 1,1 \rangle$, $m(B) = \langle 2,1 \rangle$, $m(C) = \langle 3,1 \rangle$. Clearly m is not a consistent user-to-point

mapping since

$$\sqrt{10} = d_u(A,B) \neq d_p(m(A),m(B)) = 1$$

Let's consider the different mapping m_0 shown in Figure 3.1(b). This mapping is consistent, since $d_u(A,B) = d_p(m(A),m(B))$ and $d_u(A,C) = d_p(m(A),m(C))$. Overall, there are 4 consistent mappings that are shown in Figure 3.1. ∎

Given a consistent user-to-point mapping, it is possible to compute the probability of that mapping being correct by using the knowledge BK. Intuitively, the probability is computed as the conjunction of the probabilities of each user u being in the cell where u is mapped by m. To formalise this intuition we first need to define the $up(p)$ function that returns, for each point $p \in S$, the cell g of G that contains p. Then, the absolute probability $P[m]$ that a consistent user-to-point mapping m is correct is:

$$P[pos(u_1) \in up(m(u_1)) \wedge \ldots \wedge pos(u_{|U|}) \in up(m(u_{|U|}))] \tag{3.1}$$

Since, by Definition 3.1.1, the probability that a user is located in a given cell is independent from the other users' positions, we can rewrite Equation 3.1 as shown in Property 3.1.1.

Property 3.1.1 The absolute probability $P[m]$ that a consistent user-to-point mapping m is correct is: $\prod_{u \in U} P[u, up(m(u))]$.

We can now define the probability that a user-to-point mapping m is correct relatively to the set M of all consistent user-to-point mappings. We call this distribution of probability the *relative probability* that a mapping is correct.

Property 3.1.2 The relative probability $P_r[m]$ that a consistent user mapping m is correct is:

$$\frac{P[m]}{\sum_{m' \in M} P[m']}$$

■ **Example 3.3** We continue from Examples 3.1 and 3.2. The absolute probability of m_0 is $P[m_0] = P[A,g_2] \cdot P[B,g_1] \cdot P[C,g_1] = 1/6^3$. With analogous computations, the absolute probabilities of the other three consistent mappings are: $P[m_1] = 1/6^3$, $P[m_2] = 3/6^3$ and $P[m_3] = 9/6^3$. Consequently m_3 is the most likely mapping and $P_r[m_3] \simeq 0.65$. As a result the adversary can conclude, with high likelihood, that A is located in $\langle 1,1 \rangle$, B is located in $\langle 4,2 \rangle$ and C is located in $\langle 3,2 \rangle$. ∎

From Property 3.1.2, we can also derive the probability that a user is located in a given area A, as shown in Property 3.1.3.

Property 3.1.3 The probability $P[pos(u) \in A]$ that a given user u is located in an area A is given by:

$$\sum_{m \in M | m(u) \in A} P_r[m]$$

Given the above formalisation, we can define an attack as the process of identifying consistent user-to-point mappings and of evaluating their relative probabilities using BK. An attack may be aimed at identifying the exact or approximated (in terms of an area) position of one or more specific users, or aimed at discovering the position of all users.

Modelling the attack exploiting users' clustering

The attack process presented in Section 3.1.2 has a clear computational limit due to the combinatorial explosion of the number of the mapping functions that are exponential in the number of users. For this reason, the problem of computing the consistent mappings is intractable, and not likely to have a practical solution even for small sets of users.

In order to show that there are practical and still threatening attacks, we now formalise an attack aimed at discovering the area where the user is with a looser approximation. The general idea is to cluster the users according to their distances, hence identifying sets of users close to each other, and then using BK to map clusters to geographical regions. As an example, clusters of appropriate dimensions may correspond to cities, and the adversary may be able to associate a cluster of users to the correct city as their actual approximate position.

In this attack we model an arbitrary geographical region z in S as the set of cells from G that overlap with z. We denote with Z a set of these regions. For example, the regions in Z are the cities where the adversary tries to locate users. The minimum distance between regions is defined as:

$$d_z^{min}(z_1, z_2) = \min_{g_1 \in z_1, g_2 \in z_2} d_g^{min}(g_1, g_2)$$

where $d_g^{min}(g_1, g_2)$ is the minimum distance between two cells g_1 and g_2. The maximum distance $d_z^{max}(z_1, z_2)$ between two regions is defined analogously.

In Section 3.1.5, while considering cities as the regions, we show that, given OK, it is possible to identify a set C of clusters of U that are actually composed by persons located in cities. Note that C is not a partition of U, since we are interested in a set of regions not covering the entire space (e.g., only the largest cities). After the clustering, we need to associate each cluster with the correct region, so that we can derive the geographical position of the cluster. For this purpose, we model the *cluster-to-region* mapping $m : C \to Z$ that represents an association of each cluster to a region. Conceptually, a cluster-to-region mapping is similar to a user-to-point mapping, with the only theoretical difference that in a cluster-to-region mapping all clusters are mapped to different regions, while in the user-to-point mapping two or more users can be mapped to the same point.

Intuitively, this attack becomes practically feasible when the number of clusters and regions is small, as in the case of the main cities of a country.

Analogously to the case of user-to-point mappings, we are interested in identifying the consistent cluster-to-region mappings, hence excluding the ones inconsistent with available distance information. Intuitively, a cluster-to-region mapping m is *consistent* if for each pair of clusters and each pair of users in the two clusters the distance between the two users is smaller (greater, respectively) than the maximum (minimum, respectively) distance between the two regions *corresponding* to the two clusters. Clearly the consistency condition can only be checked for those users such that $d_u()$ is defined.

Definition 3.1.3 A cluster-to-region mapping m is called *consistent cluster-to-region mapping* if, for each pair of clusters $c_1, c_2 \in C$, and for each pair of users $u_1 \in c_1$, $u_2 \in c_2$ such that $d_u(u_1, u_2)$ is defined, the following holds:

$$d_z^{min}(m(c_1), m(c_2)) \leq d_u(u_1, u_2) \leq d_z^{max}(m(c_1), m(c_2))$$

Similarly to the case of user-to-point mappings, we want to assign a probability to the

consistent cluster-to-region mappings. This probability can be derived from $nu()$. Indeed, extending Definition 3.1.1, the probability $P[u,z]$ that a generic user u is located in region z is: $\sum_{g \in z} P[u,g]$.

Since by Definition 3.1.1 the probability that a user is in a region z is independent from the probability of other users being in z and each user has the same probability, the probability $P[c,z]$ that all users in a cluster c are in z is given by: $(P[u,z])^{|c|}$.

We can now define the absolute probability of a consistent cluster-to-region mapping.

Property 3.1.4 The *absolute probability* $P[m]$ that, given a clustering C and a generic user u, a consistent cluster-to-region mapping m is correct is:

$$P[m] = \prod_{c \in C} (P[u, m(c)])^{|c|}$$

Property 3.1.5 defines the relative probability of a mapping.

Property 3.1.5 Let C be a clustering and M the set of consistent cluster-to-region mappings. The *relative probability* $P_r[m]$ that a consistent cluster-to-region mapping m is correct is:

$$P_r[m] = \frac{P[m]}{\sum_{m' \in M} P[m']}$$

3.1.3 Techniques

In this section we first show how the attack based on user clustering can be computed in practice. Then, we describe an additional step, based on trilateration, that, as shown in Section 3.1.5, significantly improves the attack performances.

Computing clusters of users

We now present a clustering technique designed to identify k groups of users that are located in user-dense areas of S. The computation of the clusters of users needs to be based only on the observation knowledge OK, i.e., on the relative distances between some pairs of users. We adopt the *Single Linkage* clustering technique that indeed relies only on relative distances [53]. The idea of this technique is to start with a set containing a single user for each cluster, and then to iteratively merge the two closest clusters. The notion of "closest clusters" is based on the minimum distance between two clusters c_1 and c_2 defined as follows:

$$d_c^{min}(c_1, c_2) = \min_{u_1 \in c_1, u_2 \in c_2 \text{ s.t. } d_u(u_1,u_2) \text{ is defined}} d_u(u_1, u_2)$$

Note that $d_c^{min}(c_1,c_2)$ is undefined if $d_u(u_1,u_2)$ is undefined for all pairs of users $u_1 \in c_1$, $u_2 \in c_2$.

The clustering procedure is presented in Algorithm 1. In the initialisation phase (Lines 1 and 2), the $|U|$ clusters are created, each one containing a single user, and the set Z of regions is ordered accordingly to the number of expected users. This number is actually derived from background knowledge that includes population density for each cell. Expected number of users can be easily derived from this. The number n of expected users in the k-th most user-populated region in Z is used in the clustering steps of the algorithm.

At each iteration, the two closest clusters are merged (Lines 4 to 6). The termination of the iteration (and consequently of the algorithm) depends on three conditions. The first condition says

Algorithm 1 *SingleLinkageClustering*

Input: the set U of users, a set Z of sets of cells, the function $du()$, an integer k.
Output: a set C of sets of users.
Procedure:

1: $C = \{\{u_1\},\{u_2\},\ldots,\{u_{|U|}\}\}$
2: Order Z in decreasing order with respect to the expected number of users. Let n be the number of users in the k-th element of Z.
3: **while** ($|C| > 1$) AND (exist $c_1, c_2 \in C$ such that $d_c(c_1, c_2)$ is defined) AND (the number of users in the k-th largest cluster of C is less than n) **do**
4: Find clusters c_1 and c_2 in C such as their distance is minimal (closest clusters)
5: $C = C \setminus \{c_1\} \setminus \{c_2\}$
6: $C = C \cup \{c_1 \cup c_2\}$
7: **end while**
8: Remove from C the sets with cardinality smaller than n;
9: **return** C;

to stop when $|C| = 1$, since in this case the users are all into a single cluster, and the algorithm just fails to identify the clusters that are associated with the regions of Z. The second termination condition is satisfied when no distance among any pair of current clusters is known. This also indicates a "failed clustering" unless the third condition is also verified. The third condition denotes a "successful clustering". When it is verified, the algorithm has identified a clustering set C containing k clusters each having a number of users at least as large as n. Intuitively, this condition is aimed at identifying the clusters that correspond to the k most populated regions. Since we are only interested in these clusters, we remove the others before returning C (Line 8).

■ **Example 3.4** Figure 3.2 graphically shows the clusters identified by Algorithm 1 after 1500 and 2500 iterations in a run with 16,000 users in France. Each dots corresponds to a user, with black dots representing users belonging to a "large" cluster (i.e., having cardinality larger than n). After 1500 iterations only 4 "large" clusters are found, while after 2500 iterations the algorithm identifies a total of 8 clusters that actually correspond to some of the 11 most populated cities of France. ■

The number of iterations of Algorithm 1 is linear in the number of users. The operation that dominates the temporal complexity of each iteration is the computation of the two closest clusters. Without any optimisation, this computation would require to calculate the distance between any pair of clusters. This operation is quadratic in the number of users, making the worst case time complexity of the algorithm $O(|U|^3)$ and the space complexity linear in $|U|$, the space needed to store C.

To improve the time complexity we designed a data structure that stores, for each cluster c: a) the set $c.users$ of users in the cluster, b) the list $c.dist$ of pairs $\langle c', d_c(c,c')\rangle$ for each $c' \in C$ such that $d_c(c,c')$ is defined, and c) the element $c.min$ of $c.dist$ having the smallest value of $d_c(c,c')$. Note that, since users' and clusters' distances are symmetric, it is not necessary to store all pairwise distances between clusters, but only half of them. In the initialisation phase it is necessary to compute $d_u()$ for each pair of users and the worst case time complexity of this

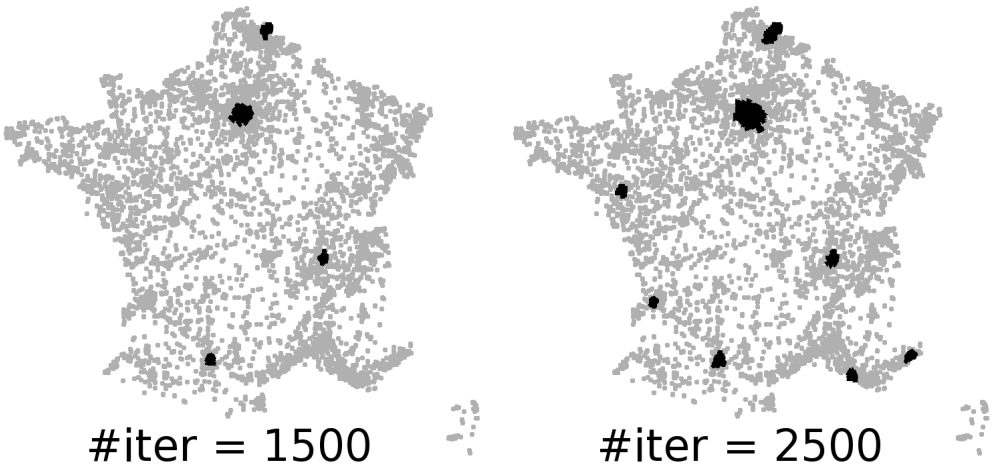


Figure 3.2: Example of execution of Algorithm 1

operation is $O(|U|^2)$. The operation that dominates the complexity of each iteration is to merge $c_1.dist$ and $c_2.dist$ for two clusters c_1 and c_2. This is analogous to compute the union between two sets. Since we define a total order among clusters and we use it to maintain $c_1.dist$ and $c_2.dist$ ordered, the union of $c_1.dist$ and $c_2.dist$ can be computed in a time linear in the sum of the number of users in c_1 and c_2, that is bounded by $|U|$. To summarise, the initialisation phase has a worst case time complexity of $O(|U|^2)$. Each iteration has a worst case time complexity of $O(|U|)$ and at most $|U|$ iterations are required, hence the worst case time complexity of the algorithm is $O(|U|^2)$.

3.1.4 Computation of consistent mappings

Once clusters of users have been identified, cluster-to-region mappings should be generated and the consistent ones selected. Two problems arise in this process. The first problem is due to what we call "cluster stretching": when the clustering algorithm is run, the large majority of the users in a cluster are located in a region of Z, but few of them are also located slightly outside the region (e.g., in the suburbs of a city). For example see Figure 3.3 where the rectangle represents the minimum bounding rectangle (MBR) of Paris and the dots (both black and grey) are the users in a cluster. It can be observed that, while the great majority (90%) of dots are within the MBR, few of them are slightly outside.

The "cluster stretching" problem could lead to erroneously classify a mapping as inconsistent. To face this problem, we introduce a tolerance error factor $\alpha \in (0,1)$ in the condition to check the consistency (see Definition 3.1.3) as follows:

$$\alpha \cdot d_z^{min}(m(c_1), m(c_2)) \leq d_u(u_1, u_2) \leq \frac{1}{\alpha} \cdot d_z^{max}(m(c_1), m(c_2))$$

The intuition is that, when α is close to one, the tolerance is low. For values of α closer to 0, the value of the left part of the inequality gets smaller, while the right part gets larger,

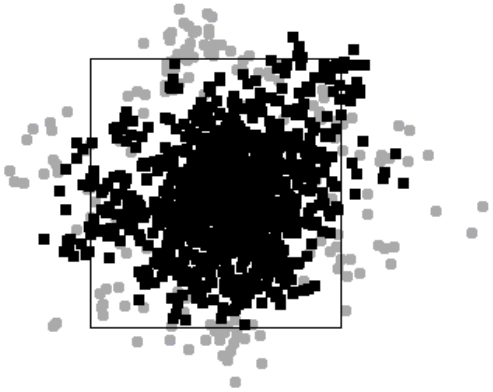


Figure 3.3: The "cluster stretching" problem.

hence making the condition easier to satisfy even if a user may be slightly outside the region. In Section 3.1.5 we show the impact of the tolerance error factor and we motivate the choice of the value used in our experiments.

The second problem is related to the fact that the number of possible cluster-to-region mappings is the same as the number of all ordered sequences of $|C|$ distinct elements of Z, i.e., $|Z|!/(|Z|-|C|)!$. Clearly, this leads to a large number of possible mappings, also for small sets Z and C. Since for each possible mapping we need to check if that mapping is consistent, it is necessary to optimise this operation. For this purpose the minimum and maximum distance between each pair of regions in Z (i.e., $d_z^{min}(z_1,z_2)$ and $d_z^{max}(z_1,z_2)$) are pre-computed off-line. We also pre-compute, for each pair of clusters $c_1,c_2 \in C$, the minimum ($d_c^{min}(c_1,c_2)$) and maximum ($d_c^{max}(c_1,c_2)$) distance between each pair of users $u_1 \in c_1$ and $u_2 \in c_2$. It follows that checking if a mapping is consistent requires evaluating the following two inequalities for each pair of clusters c_1, c_2 of C.

$$\alpha \cdot d_z^{min}(m(c_1),m(c_2)) \leq d_c^{min}(c_1,c_2)$$

$$d_c^{max}(c_1,c_2) \leq \frac{1}{\alpha} \cdot d_z^{max}(m(c_1),m(c_2))$$

By using this optimisation, the process of checking if a mapping is consistent has a worst case time complexity of $O(|C|^2)$. Considering that we need to run this computation for each possible mapping, the computational complexity is $O\left(|C|^2 \cdot \frac{|Z|!}{(|Z|-|C|)!}\right)$. Clearly this may be still problematic in general, however, as we show in Section 3.1.5, the attack can be effectively computed for values of $|C|$ and $|Z|$ that are sufficient to identify the approximate position of the users located in the most populated cities.

Evaluation of probabilities

The last step of the attack process consists in computing the absolute and relative probabilities of each consistent cluster-to-region mapping. This is done accordingly the formulas presented in Section 5.2.1. We compute the function $nu(g)$ from the distribution of inhabitants of g, which is public information [O5], and by assuming that the users of the service are uniformly distributed

in the population. Formally, given $inhab(g)$ the number of inhabitants of g and $totPop$ the total population in the spatial domain S: $nu(g) = inhab(g)/totPop \cdot |U|$.

While, in theory, the number of consistent cluster-to-region mappings can be in the same order of all the possible cluster-to-region mappings, in practice these mappings are a small fraction (in our experiments, in which major cities are considered as regions, they are in the order of a few units at most).

A technical difficulty arises in the computation of the absolute probability of a mapping due to the fact that some of the values used in the computations may be non-negative numbers very close to 0 (for example, these values can be in the order of 10^{-10000}). The precision of standard floating point representations (e.g., primitive double type in Java) is not sufficient to manage these values. We solved the problem with non approximated decimal representations and, in particular, with the "BigDecimal" standard Java class, that, however, has a significant negative effect on the computation time.

Trilateration

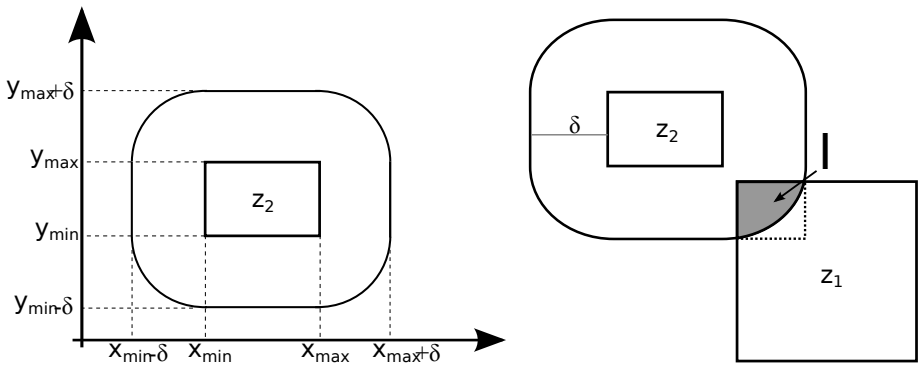
When the cluster-to-region mapping with highest probability is found, the adversary learns the approximate position of the users belonging to each cluster. We now present a technique that can lead the adversary to obtain more precise location information for these users and to also compute the approximate location of the other users. The technique is based on trilateration [24]. It takes as input the approximate locations given by the candidate cluster-to-city mapping with highest relative probability.

Our solution addresses two differences with respect to "standard" trilateration: (1) only approximated location information is available so we have to trilaterate with rectangular areas instead of points; (2) approximate location of users belonging to a cluster can be incorrect, due to the "cluster stretching" problem. We tackle this inaccuracy through the "refinement" technique.

(1) To address the difficulty of having only approximated location information, we represent known users' positions as rectangular regions and we operate geometrical intersections to locate users. Intuitively, suppose that a user u_1 is at distance δ from another user u_2 located in a city z_2. This bounds the location of u_1 to the set of points whose minimum distance from z_2 is at most δ. We denote this "extended" region as $ext(z_2, \delta)$, see an example in Figure 3.4(a). If the (approximate) position user u_1 is unknown, then the adversary learns that u_1 is located inside $ext(z_2, \delta)$. Otherwise, if the adversary already knows that u_1 is located in z_1, it can be deduced that u_1 is located in the intersection between z_1 and $ext(z_2, \delta)$ (e.g., the dark region in Figure 3.4(b)).

When the above geometrical computation reduces the uncertainty about the position of a user u, this restriction can "propagate" to all the users u' whose distance from u is known. This idea leads to our iterative Algorithm 2 that computes the function r associating a spatial region to each user. In the initialisation phase (Lines 1-4), each clustered user is assigned to the corresponding city, while the other users are assigned to the entire space. At each iteration (Lines 6-8) for each pair of users $\langle u_1, u_2 \rangle$ whose distance is known, we compute the intersection between the approximate position of u_1 and $ext(r(u_2), d_u(u_1, u_2))$ where $r(u_2)$ is the approximate position of u_2. The computation terminates when there are no more approximate positions to restrict.

Note that the intersection between the approximate position of u and $ext(r(u_2), d_u(u_1, u_2))$ is not always a rectangle (see again Figure 3.4(b)). When iterating, this can lead to figures with complex shapes that require a non-constant-time computation of the geometrical intersection



(a) $ext(z', \delta)$ (b) Geometrical intersection

Figure 3.4: Computation of intersection

function. In order to ease the computation, we first compute the exact intersection and then we bound it with its MBR (dotted lines in Figure 3.4(b)). In our experiments this optimisation leads to a negligible decrease in the effectiveness of the algorithms (in terms of the size of the approximated positions) but it decreases significantly the computation time.

Algorithm 2 *Trilateration*

Input: the set U of users, a set C of clusters, a cluster-to-region mapping $m : C \to Z$, the function d_u.
Output: the function $r(u)$ for each user $u \in U$.
Procedure:

1: **for all** $u \in U$ **do**
2: **if** $u \in$ cluster $c \in C$ **then** $r(u) = m(c)$
3: **else** $r(u) =$ entire space
4: **end for**
5: **while** (at least 1 region is reduced) **do**
6: **for all** pairs of users $\langle u_1, u_2 \rangle$ such that $d_u(u_1, u_2)$ is defined **do**
7: $r(u_1) = MBR(r(u_1) \cap ext(r(u_2), d_u(u_1, u_2)))$
8: **end for**
9: **end while**
10: **return** function $r()$

(2) The "cluster stretching" problem (see Section 3.1.4) can lead to erroneous associations between users and regions. In our experiments we observe that the "cluster stretching" problem can drastically affect the precision of Algorithm 2. Intuitively, this is caused by the iterations of Algorithm 2 that propagate errors. To prevent this, we adopt a solution that improves the precision of the cluster-to-region mapping and hence, as shown in Section 3.1.5, drastically increases the precision of our attack. The idea is that, after computing the mapping with highest probability, we "refine" the clustering so that each cluster contains a fraction of the expected number of users in the corresponding city. The refinement of a cluster c takes as input a number n' and the set

of users belonging to c: these users are clustered together until a cluster with exactly n' users is found. This cluster represents the refinement of c.

The refinement increases the precision in terms of the correct association of users to a region since, in general, the last users added to a cluster have a higher probability to be outside the region, as shown in Example 3.5. We use the refinement to restrict the set of users in a cluster to a fraction of the expected users in the corresponding region. We call "refinement factor" the percentage of expected users that are discarded when refining the cluster. For example, with a refinement factor of 20%, a cluster is refined including 80% the expected users in the corresponding region.

■ **Example 3.5** Consider Figure 3.3 again. The expected number of users in Paris is 1200, while 1310 users have been clustered into c. The figure shows with black dots the 1200 users in the refinement of c (with refinement factor of 0%), while the grey dots represent the other 110 users that are in c but not in its refinement. The percentage of users of c that are located in the MBR is about 90% while considering the refinement of c the percentage is about 94% and among the 110 users in c but not in the refinement the percentage is about 45%. This clearly indicates that the first users that are clustered together actually have a much higher probability of being within the MBR. ■

3.1.5 Experiments

Our experiments simulate the observation knowledge (OK) that an adversary may obtain by running a friend-finder service in which privacy protection is implemented through a distance preserving transformation. We use as background knowledge (BK) publicly available geographic data and population density. The aim of the attack is to infer the actual position of users (supposed to be hidden to the service) in terms of a region in which they are located. In Section 3.1.5 we present the experimental setting. We evaluate the effectiveness of the attack by first focusing on the clustering technique (Section 3.1.5) and then showing how trilateration improves the attack (Section 3.1.5). Finally, in Section 3.1.5 we discuss other experimental results we obtained.

Experimental setting

The overall structure of our experimental evaluation is the following: we first simulate the position of some users in a geographical area. Then, we compute the distance between some pairs of users (hence simulating OK) and we use this data to perform the attack. Finally, using the original information about users' position, we evaluate the correctness of the attack.

The simulation of users' position consists in randomly choosing a point in the geographical area according to a probabilistic distribution. We consider three scenarios corresponding to the geographic areas covering Australia, France and the entire world. The intuition behind the choice of the first two scenarios is that the adversary could know, from a number of different sources, the country where users are located. In the world scenario we assume the adversary does not have this knowledge. The choice of using France and Australia scenarios is due to the fact that the two countries have very different average population density, (about 2.8 $inhabitants/km^2$ in Australia, about 116 $inhabitants/km^2$ in France) and also large difference in population density variance (Australia is characterised by large unpopulated regions, while in France the variance of population density is much lower). Differently from what expected, we obtained very similar results for these two scenarios.

The probabilistic distribution we adopted is taken from GPWv3 [O5], a dataset that provides density information by dividing the world into cells and providing the number of inhabitants for each cell. In our experiments we observed that smaller cells result in more effective attacks while not significantly impacting on the computation time. Hence, we use the cells at the highest available resolution, i.e., 2.5 arc-minutes, that corresponds to a cell edge of about $5km$ at the equator. Since no public information is available at a higher resolution, in the current setup we assume that, within each cell, the population density is uniform. In the following we denote with "#users" the value $|U|$.

Another parameter that has an impact on the attack is the number of distances between users that are known to the adversary. In the friend-finder service, this corresponds to the mean number of friendships per user ($\#friends$). In order to capture the intuition that users tend to have more friends in a close-by region, 50% of each users' friends are located in an area whose size is 2% of the simulated region. In the case of France this corresponds to a distance of 100km from the user.

The parameters k and $|Z|$ are fixed to 8 and 11, respectively (note that, when the clustering is successful, $|C| = k$). In our experiments (not reported in the following) we observed that while these values already enable powerful attacks, higher values of these two parameters would improve it even more. However, higher values negatively impact on the performances, due to the reasons explained in Section 3.1.3. Using these two values for k and $|Z|$ and the default values for $\#users$ and $\#friends$, we can compute an attack in about 2 minutes on a 2.26GHz CPU with 4GB of main memory.

We evaluate the attack in three scenarios by varying parameters, showed in Table 3.1, with default values in bold. The value of the tolerance error factor α is set to 0.75, empirically chosen, as will be shown in Section 3.1.5. The results are computed as the average, as well as minimum and maximum, out of 10 runs.

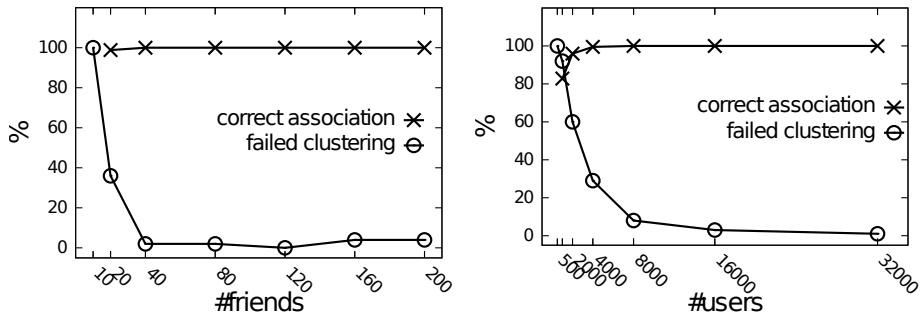
Table 3.1: Parameter values

Parameter	Values
#users	500, 1000, 2000, 4000, 8000, **16000**, 32000, 64000
#friends	10, 20, 40, **80**, 120, 160, 200
refinement	0, 20, 40, 60, **80**, 90, 92, 94, 95, 96, 97, 98, 99
α	1, 0.95, 0.90, 0.85, 0.80,**0.75**, 0.7, 0.65, 0.6, 0.55, 0.5
Scenario	Australia, France, World

Evaluation of the Attack Based on Clustering Only

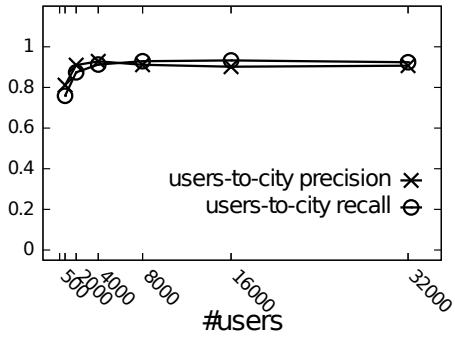
The aim of the experimental evaluation of the clustering technique is to assess the attack effectiveness in terms of the "cluster-to-city association" and "users-to-city association".

The "cluster-to-city association" measures the percentage of clusters that are correctly associated to the corresponding city. A cluster is considered correctly assigned to a city if at least 50% of its users are located within the assigned city's MBR. We can observe from Figure 3.5(a) and from Figure 3.6(a) that, for values of $\#friends$ equal to or larger than 40 in France and equal or

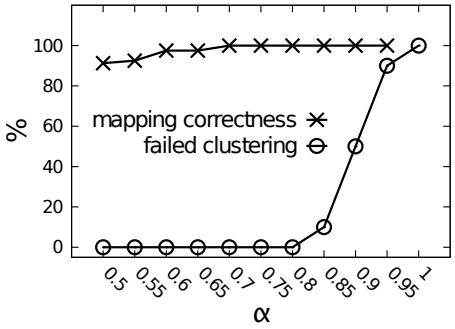


(a) Cluster-to-city association in France

(b) Cluster-to-city association in France



(c) Users-to-city association in France

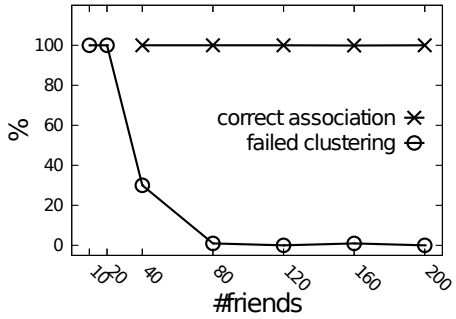


(d) Cluster-to-city association in France

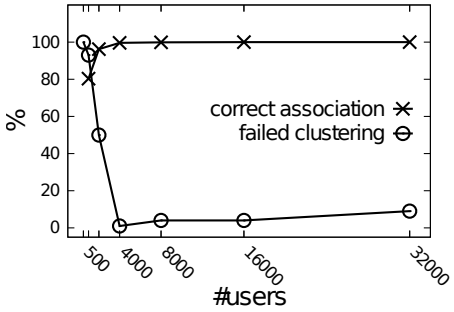
Figure 3.5: Results obtained without trilateration in France

larger to 80 in Australia the clustering algorithm fails less than 3% of the times while, when it does not fail clustering, the cluster-to-city association is correct 100% of the times. This is an important property of our attack: when the number of users is sufficiently large (in this case we are using the default value of 16000) if the clustering does not fail, then the adversary knows with high likelihood that the cluster-to-city association is correct. This can also be observed in Figure 3.5(b) and in Figure 3.6(b) showing that the cluster-to-city association is always correct for large values of *#users*. Vice versa, for *#users* smaller than 4000, we observe that a wrong cluster-to-city association is more likely. Australia and France scenarios show a similar trend, both varying the number of users and the number of friends: however we can observe that, compared to France, in Australia an higher number of friend, but a lower number of users is needed to avoid failed clustering. This can be motivated observing that the population of Australia is mostly located in the major cities, while in France the users are more distributed in all the country: therefore more connections are needed to link users from city to city, but since most of them are located within them the number of users can be lower with respect to France.

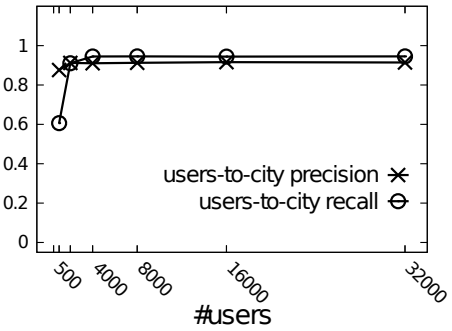
We observe a similar trend in the world scenario (see Figure 3.7(a)). In this case the algorithm always returns correct cluster-to-city associations when there are at least 8000 users. Compared to the France scenario, the percentage of failed clustering decrease more slowly for larger values of *#users*. Indeed, the clustering fails about 20% of the times with 64000 users. This is due to the fact that in the world scenario we are simulating a much smaller percentage of the population.



(a) Cluster-to-city association in Australia

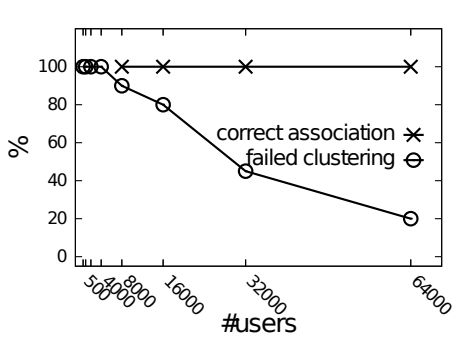


(b) Cluster-to-city association in Australia

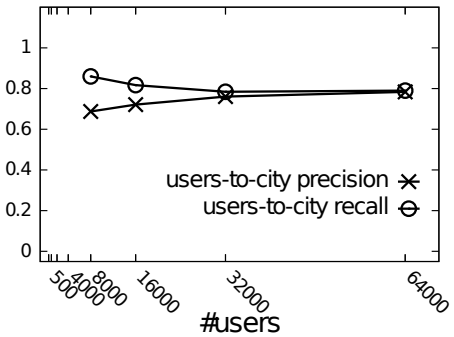


(c) Users-to-city association in Australia

Figure 3.6: Results obtained without trilateration in Australia



(a) Cluster-to-city association in the world



(b) Users-to-city association in the world

Figure 3.7: Results obtained without trilateration in the world

Indeed, with 32000 *#users* in France we simulate about the 0.04% of the population, while with 64000 *#users* we simulate only the 0.00001% of the world's population.

The "user-to-city association" measures the precision and recall of the association between a user and a city. In this case, the association of a user with a city is correct if the cluster containing that user is associated to the city where the user is actually located. Since, as we observed above, the cluster-to-city association is always correct for sufficiently large values of *#users*, the

user-to-city association helps us understanding the impact of the "clustering stretching" problem. Consider Figure 3.5(c) and Figure 3.6(c): in both France and Australia for values of #*users* equal to or larger than 2000 the average precision is almost not affected by the value of #*users* and it is always above 90%. Similar trend and values can be observed for the recall. This means that most of the users that are located in a city are actually reported as being in that city and that, in most of the cases, a user reported being in a city is actually located there. A very similar result can be observed for the world scenario (see Figure 3.7(b)). The main difference is that we need a minimum of 8000 users, since for smaller values of #*users* the clustering always fails.

One experiment is devoted to find a proper value for the tolerance factor α described in Section 3.1.4. As shown in Figure 3.5(d), the cluster-to-city association often fails for large values of α (above 0.85) since no consistent mapping is found. For values smaller than 0.65, more consistent mappings need to be evaluated, resulting in much higher computation time and higher probabilities of errors in choosing the correct mapping. A good trade-off is found for values of α between 0.7 and 0.8, for which there are no failed clusterings and the cluster-to-city associations are always correct. For these reasons, in our experiments we used a value of $\alpha = 0.75$.

Evaluation of the attack with trilateration

The next set of experiments evaluates the effectiveness of the trilateration step, in terms of its correctness and of the percentage of users that it can associate to a "small" area.

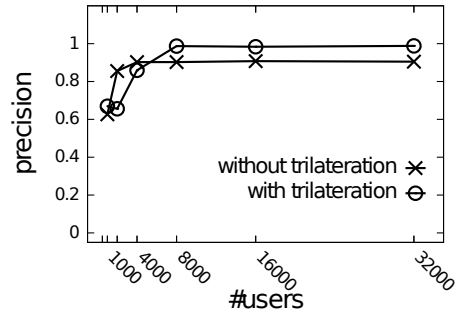
Figure 3.8(a) and 3.8(b) highlight the improvement of the attack due to trilateration. Figure 3.8(a) shows that the trilateration technique improves the precision of the attack when the users' number is greater than 4000. This is mainly due to the refinement step (see Section 3.1.4). At the same time, Figure 3.8(b) shows that, when trilateration is used, it is possible to correctly associate more than 60% of the users to an area smaller than $50km^2$. The attack with trilateration is about three times better along this metric than the attack without trilateration that can only locate the users in the cities. The results in the world scenario (not shown here) are very similar, given that the number of users is sufficiently large.

Figure 3.8(c) helps us understanding the size of the regions where the attack locates the users. Approximately 40% of the users are correctly associated with a region smaller than $25km^2$. The attack can also discover a more precise location (less than $5km^2$) for about 416 users on average (i.e., 2.6% of the users). A few users (5, on average) can be localised in an area smaller than $1km^2$.

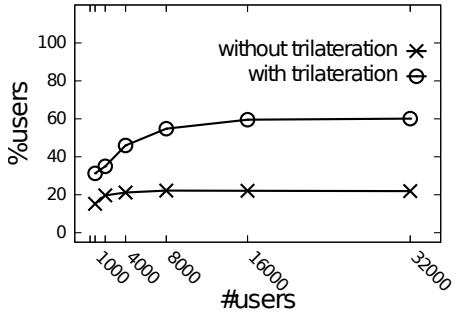
Figure 3.8(d) and Figure 3.8(e) show the percentage and the precision, respectively, of users located in an area smaller than $50km^2$ for different values of the refinement factor. From the comparison of the two figures, we can observe that there is a trade-off between precision and the percentage of users whose location is discovered: increasing the refinement with values higher than 80% makes the precision converge to 1, but the percentage of users whose location is discovered decreases. Experiments show that the threshold value depends on the value $\#friends$: when this value is equal to 20, better results are obtained with a refinement factor of 0, while if $\#friends$ are more than 80 a refinement factor of the 80% is needed.

Other experimental results

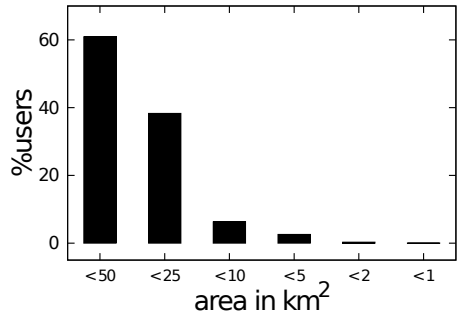
We run a set of experiments to evaluate the effectiveness of our attack when only short distances (less than $3km$) are known to the adversary. This is the situation that would result when using a



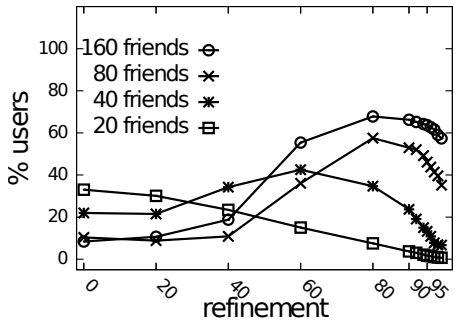
(a) Precision of positioning users in an area smaller than 50 km^2 in France



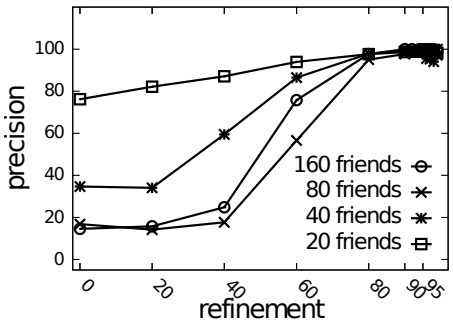
(b) Users correctly positioned in an area smaller than 50 km^2 in France



(c) Size of the regions where users are correctly positioned



(d) Users correctly positioned in an area smaller than 50 km^2 in France



(e) Precision of positioning users in an area smaller than 50 km^2 in France

Figure 3.8: Results of the attack with trilateration

defence like the one proposed in [22]. We adapted our attack to this situation by changing the clustering algorithm so that it terminates when the distance between the two closest clusters is larger than 3km. A major difference is due to the fact that in absence of long distances it is not possible to compute if a cluster-to-region mapping is consistent or not. For this reason we skip the consistency check and compute the relative probability for each mapping. This solution incurs in two distinct problems. The first is a computation cost problem: computing the relative probability among all the possible mappings is much more time consuming than first selecting consistent

mappings and then computing their probability. For this reason we had to reduce the values of k and $|Z|$ to a maximum of 3 and 4, respectively. The second problem is related to the fact that, by assuming that all mappings are consistent, it is much more frequent that an incorrect mapping has a high relative probability, hence resulting in an incorrect cluster-to-city association. Despite these problems, our results show that the attack is still effective in most of the cases. Using default values for $\#users$ and $\#friends$, $k = 2$ and $|Z| = 4$ in France, our clustering algorithm has always been able to correctly compute the clusters, and the cluster-to-city association always resulted to be correct. The computation time is in the order of 2 minutes. However, we observed that, differently from the case in which even long distances are known to the adversary, increasing the values of k and $|Z|$ can result in worse performance, since wrong cluster-to-city associations are more frequent. For example, by using $k = 3$ and $|Z| = 4$ we obtained 73% of correct cluster-to-city associations. Overall, we can conclude that the application of a defence technique like the one proposed in [22] to our reference scenario (e.g., a friend-finder service) does not protect from our attack, that it still able to identify the correct position of many users. On the other hand this defence poses some challenges to our attack both from the computational point of view, and from its effectiveness, since it reduces the number of possible users whose position can be directly identified.

3.1.6 Summary

In this section we investigated how distance information between moving individuals can be used to violate location privacy in presence of background knowledge about population density. In contrast with other kinds of background knowledge that have been considered in the literature we base our attack entirely on public knowledge and on realistic partial distance information as it could be released by a Geo-Social Network friend-finder service. Our experiments show that the knowledge of a relatively low number of distances is sufficient to correctly position the majority of users in an area smaller than 50 km^2 and to correctly identify the position of some users in an area smaller than 1 km^2.

We believe that this study shades a new light on the safety of distance preserving transformations as privacy techniques; at the very least, it provides new important elements to design safer defence techniques considering public knowledge on population density.

3.2 Co-location privacy

Through Geo-Social Networks it is possible to publish posts, photos, check-ins together with the locations in which they were generated. Location privacy and absence privacy in these kind of social networks have already been discussed thoroughly in [14]. We now want to consider the co-location privacy issue: a co-location occurs when a user can be located with another user in a given place at a given time.

In some cases being co-located with another cannot be a problem, while in other does: for instance if Alice is Dave's ex- girlfriend and her actual boyfriend sees two posts linking them to the same place, he can be jealous and start a quarrel with Alice. Or, in another scenario, if Alice is known as a corrupted person and Dave is a politician, a malicious journalist, seeing two posts linking them to the same place, can make a scoop saying that there was a meeting between the

two, compromising Dave's career. A user can have many other reasons for not wanting that a specific co-location occurs, but what is undeniable is that there is an underlying privacy issue. In this section we'll analyse and formalise this threat.

3.2.1 Classification and preliminar definitions

The co-location privacy concerns can stem in many location based services. For instance in publication of traces or continuous positions tracking. Among all the possible scenarios in which the co-location privacy issue can arose, we consider as the reference one a Geo-Social Network.

This choice implies that we assume that users do not publish continuously their positions, but only in a sporadic way and, more commonly, in places that are unusual for the users. This means that the adversary do not know, for each user, the home/work pairs locations or habits connected to the everyday routine: we'll discuss later in the chapter how this assumption will affect the co-location threat formal definition.

In a Geo-Social Network each user has the possibilities to publish a post enhanced with the geographical location and indicating a (possibly empty) set of other users with him. Each published post, or resource, therefore has an author (the one that is publishing the post), a content (a text, a photo, video, etc.), a geographical position, a temporal component (typically the time of the post's publication) and a set of users included in the post. More formally:

Definition 3.2.1 Given a user u, a resource r_u is a tuple: $< Udata, STdata, Content >$ where

- r.Udata: set of identifiers of users, included the author u
- r.STdata: r.Sdata spatial component, r.Tdata temporal component
- r.Content: resource itself

and u is the author of the resource.

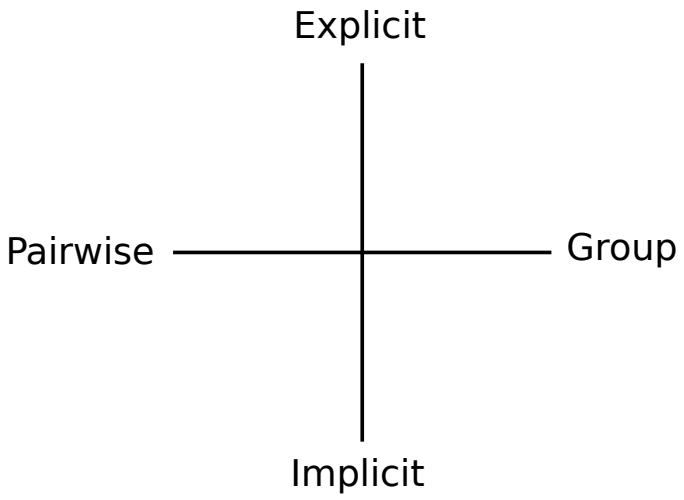


Figure 3.9: Classification of co-location threats

A co-location privacy breach can be classified depending on the number of the users involved in the co-location and on the number of resources needed to infer the co-location.

Considering the number of users involved we classify the co-location threat in pairwise or group co-location. We have a pairwise co-location threat if a user u is co-located with a user u' with whom he doesn't want to. It can happen that u have no problem of being co-located with the users u' and u'' but he does not want to be co-located at the same time with both u' and u''. If this happen we have a group co-location threat.

Instead, if we use as the classification criteria the number of resources involved, we can divide the threats in explicit and implicit. An explicit co-location threat occurs when there is a single resource that locate u together with an undesired user u'. Otherwise, if two or more resources are needed to infer the co-location threat, the co-location is qualified as implicit. As shown in Figure 3.9, we can observe that the classification based on the number of resources is orthogonal to the one based on the number of users.

In Section 3.2.2 we'll examine first the pairwise co-location threat, considering first the explicit case and then the implicit, while in Section 3.2.3 we'll discuss about group co-location. Further examples to better clarify the implicit/explicit, as well as the pairwise/group, concepts are provided in the following.

3.2.2 Pairwise co-location

In this section we consider the case in which a user does not want to be co-located together with another single user. To define the privacy threat, firstly we need to specify, for each user u, a set of rules that formally specifies which users are "undesired" for him. For this purpose, we define a *privacy preference*:

Definition 3.2.2 A pairwise privacy preference ϕ_u for the user u is the set of users $\phi_u = \{u_1, u_2, \cdots u_n\}$ with whom the user u does not want to be co-located.

■ **Example 3.6** If Alice does not want to be co-located with Bob and Dave, then her pairwise privacy preference will be: ϕ_{Alice} ={Bob, Dave}. ■

Please note that a pairwise privacy preference can be more specific if we include for each user u_i in ϕ_u the set of locations S and the temporal intervals T in which u does not want to be co-located with u_i. Moreover, u can consider a privacy breach being co-located with u_i in an area A (not only in the same place): for this purpose for each user u_i belonging to ϕ_u it is possible to include a minimum distance d within which u does not want to be co-located. The resulting privacy preference is a set of tuples, one for each user with which u does not want to be co-located:

$$\phi_u = \{\{u_1, S_1, T_1, d_1\}, \{u_2, S_2, T_2, d_2\}, \cdots \{u_n, S_n, T_n, d_n\}\}$$

Let's see an example to better clarify S, T and d.

■ **Example 3.7** Let's recall the privacy preference ϕ_{Alice} in Example 3.2.2. From ϕ_{Alice} we know that Alice does not want to be co-located with Bob and Dave. Let's suppose that Alice is a colleague of Bob: in this case it's normal that she is co-located with him during working hours, but to avoid her boyfriend's jealously she does not want to be co-located with him during the night in the pub or in the discotheque. Furthermore, Alice does not want to be co-located with

Dave within at least $200m$, since he has a very bad reputation that can affect Alice's career. Then her privacy preference will be:

$$\phi_{Alice} = \{\{Bob, \{Discotheque, Pub\}, 19:00-05:00, 0m\}, \{Dave, Everywhere, Anytime, 200m\}\}.$$

■

Even if S, T, d can better describe the co-location preference for user u,without any loss of generality, from now and in the rest of the chapter we will use the simplified version provided in Definition 3.2.2. All the definition and theorems in the following can be easily extended to meet the privacy preference with S, T and d.

For each user u_i we have now a pairwise privacy preference. Now we want to answer the question: how can a co-location privacy breach occur? In other words we are interested in identifying the circumstances under which user's privacy is at risks, starting from the naive case (explicit co-location) and studying then the more sophisticated implicit pairwise co-location threat.

Explicit

Intuitively, an explicit pairwise co-location threat occurs when there is a published resource r that locate together a user u and another user u' with which u does not want to be co-located, i. e. $u' \in \phi_u$. Please note that in this case it is not important who is the publisher of the resource r, it can be either u' or another user u''. In both cases, there is a privacy breach for u. Formally:

Definition 3.2.3 An explicit pairwise co-location threat for the user u, with pairwise privacy preference ϕ_u, is the publication of a resource r such that:

$$(u \in r.Udata) \wedge (r.Udata \cap \phi_u \neq \emptyset)$$

■ **Example 3.8** Let's assume that Alice is using a Geo-Social Network and she is very concerned about her privacy. As in Example 3.6, her privacy preference ϕ_{Alice} shows that she does not want to be co-located with Bob and Dave. Her friend Carl publish a post in the Geo-Social Network, writing "Having a drink @Pub - with Alice and Bob": we can easily see that the publication of this post is threatening Alice's privacy, since it violates her privacy preference ϕ_{Alice}. ■

The detection of an explicit pairwise co-location threat is straightforward, since the observation of a single resource is enough to state, without any doubt, if there is a threat for the user's privacy or not. In the next paragraph we'll see how different, and more complicated, is the approach in identifying the implicit pairwise co-location threat.

Implicit

In the previous section we have seen how to detect an explicit co-location threat. What if the co-location threat should be inferred by two or more resources? Let's consider an example:

■ **Example 3.9** Let's assume that Alice is in a pub with her friend Barbara. Barbara updates her Geo-Social Network status publishing the post "Having a nice evening with Alice" at 9 pm @Pub. Also Carl arrives in the same pub, meets Dave and updates his status, writing "Nice to unexpectedly see my buddy Dave" at 9.05 pm @Pub.

In this example a possible co-location occurs between all the persons mentioned. Even if neither Alice nor Dave published anything, a person that have access to both the posts of Barbara and Carl can infer that Alice and Dave were in the same place, nearly at the same time, and it is very likely that they met and they possibly have talked to each other. If Alice's privacy preference includes Dave or Carl then an implicit privacy breach can occur. ∎

Please note that inferring the implicit co-location with certainty is possible only if the two resources are published in the same place at the same time. In the example above, for instance, we cannot be 100% sure that Alice was in the pub when Carl and Dave arrived, since in the 5 minutes interval that separate the two posts Alice might be out of the pub and have gone elsewhere. On the other hand, it is highly likely that Alice was in the Pub at 9.05 since only 5 minutes before she was hanging around there with Barbara.

Hence, to deal with implicit co-location privacy we formalise the co-location threat as a probability.

Let's assume that the publication of a resource r links the user u in a location l at time t. Given u, l and t, the user's probability to be in another location l' after a time t' is defined by a function f. The value of this probability, computed through the function f, depends on the activity the user u was performing in r: for instance if in the resource r the user u is drinking in a pub with a friend, then it is very likely that after 5 minutes he will be still there. Conversely, if in r the user u is passing near the pub, driving to another location, then the probability that he will be in the surroundings after 5 minutes is much lower or null.

What if the known locations of the user u are frequent in time? Clearly the probability of u of being in a specific place l' at time t' can depend on more than one known location. A set of locations close in time can define a trajectory for a user: with this kind of data it is possible to infer the means of transport, habits and to estimate future locations as well. Hence, in this scenario, given a location l', the probability that user u is located there depends on a set of known locations in given times. Formally:

Definition 3.2.4 Given the set $KL_u = \{(l_1,t_1),(l_2,t_2),\cdots,(l_n,t_n)\}$ where each pair (l_i,t_i) represents a known location l_i at the time t_i for the user u, the probability that u is located in l' at time t',:

$$p[loc_u(t') = l' | KL_u] = g(l',t',KL_u)$$

As we discussed in Section 3.2.1, we are not taking into account such a scenario since we are assuming only sporadic publications: this implies that the known locations in KL_u are too scattered to infer habits or common locations. This is the reason underneath the fact that to compute the probability that a user u is located in l' at time t' we consider only the known location at the time closer to t'.

We now want to model the probability that two users are close in a given time t. To be close is clearly semantically different than being in the same exact place: however if the distance between two users is small enough we can approximate the concept of being co-located with the one of being close. For instance, in the recurring example in which Alice's boyfriend Carl is jealous, if Alice and Bob are not precisely located in the same place, but their positions are at a distance of $50m$ probably he is going to be jealous anyway. For this purpose we define a minimum distance

D: if two users are located within they are *close*, otherwise they are not. Please note that D can be either a system parameter or a user preference. In the latter case, a user u can specify a value for D that apply to all the users in his privacy preference ϕ_u or he can decide different values of D, one for each user $u_i \in \phi_u$.

Independently on how it is chosen, let's assume a distance value D and let's compute the probability that two users are close in a given time t. Since we consider each users's probability as independent, fixed two locations l and l', the probability that u and u' are close is the product of the probability of user u to be located in l at time t multiplied by probability of user u' to be in located in l' at the same time t, only if the distance between l and l' is not greater than D (otherwise the probability of being close is clearly zero). Since we are interested in the probability of the two users to be close everywhere, and not only in fixed locations, the overall probability should be computed on all possible locations l and l' of the spatial domain S.

To compute this probability, we first need to define a piecewise function that takes two locations as input: if the distance[3] between the two locations is smaller than the given threshold D then the function value is equal to 1, while, in all other cases, its value is equal to 0.

Definition 3.2.5 Given a distance threshold D and two locations l and l', the *closeness* function $c_D(l,l')$ is equal to:

$$c_D(l,l') = \begin{cases} 1 & \text{if } d(l,l') \leq D \\ 0 & \text{otherwise} \end{cases}$$

where function $d(a,b)$ computes the distance between locations a and b.

We can now formally define the *closeness probability* for u and u' at time t:

Definition 3.2.6 Given the users u and u', the probability $Pclose_D(t,u,u')$ that u and u''s locations are close at a given time t is:

$$\begin{aligned} Pclose_D(t,u,u') &= p\left[c_D(loc_u(t), loc_{u'}(t)) = 1\right] = p\left[d(loc_u(t), loc_{u'}(t)) \leq D\right] = \\ &= \iint_S p\left[loc_u(t) = l\right] \cdot p\left[loc_{u'}(t) = l'\right] \cdot c_D(l,l') \mathrm{d}l \mathrm{d}l' \end{aligned}$$

Now that we have the probability that two users are close at a given time t, we can model the implicit co-location threat. Intuitively we can say that there is a co-location when the probability that two users are close is very high (since we are almost sure that the users are in nearly the same place). In other words, given a resource r in which the user u is included, if it exist another published resources r' including a user u' and the probability that both u and u' are located within a distance D at time t is greater than a given threshold $\mathscr{K}$, then a co-location threat can occur if u' is included in u's privacy preference. Formally,

Definition 3.2.7 Given a real number $\mathscr{K} \in [0,1]$ and a distance value D, an implicit pairwise co-location threat for the user u, considering ϕ_u, is the publication of a pair of resource r and r' such that:

[3] The distance can be either the euclidean distance or Manhattan distance or other

1. $r \neq r'$
2. $(u \in r.Udata) \wedge (\exists u' \in r'.Udata \cap \phi_u)$
3. $\exists t \in T$, s.t. $Pclose_D(t,u,u') \geq \mathscr{K}$

where T is the temporal domain.

The above definition can be not easy to compute and analyse since the probability that u is close depends on the function f. Indeed, this function can be deliberately complex, non continuous, non monotonic, etc. We now want to study three simple scenarios (uniform distribution, bivariate normal distribution and discrete spatial domain) that for sure are non realistic but they can help us to better understand the implicit co-location threat identification problem.

Uniform probability

We assume now a naive scenario in which there are no spatial constraint(roads, lake, etc.) and all the users' move independently in the space with a maximum speed v_{max}. Therefore, given a user u and his known location l at time t, u's approximate location in a given time t' is the circular area with centre l that is reachable in the interval $t'-t$ assuming a maximum speed v_{max}. Formally:

Definition 3.2.8 Given a user u the *uniform* probability that u is located in l' at time t', given that u is located in l at time t, is equal to:

$$p\left[loc_u(t') = l' | loc_u(t) = l\right] = \begin{cases} 0 & \text{if } d(l,l') > v_{max} \cdot |t-t'| \\ \frac{1}{\pi[(t-t')\cdot v_{max}]^2} & \text{otherwise} \end{cases}$$

■ **Example 3.10** Let's refer to Figure 3.10. The user u's approximate location at time t is the circular are c_u: outside of c_u, u's probability of being there is null. The area c_u is computed from u's known position (x_u, y_u) in in a time t' earlier than time t, according to Definition 3.2.8. The borders of c_u are the furthest locations that u, starting from (x_u, y_u), can reach in the time interval $t-t'$ moving with speed v_{max}. Analogously, $c_{u'}$ is the approximate location for the user u' knowing that he was in $(x_{u'}, y_{u'})$ at time t''. Please note that the area of $c_{u'}$ is bigger than c_u: since the speed v_{max} is the same for both the users, this means that the known location for the user u' was taken in a time $t'' < t' < t$. ■

We now want to compute the probability that two users, u and u', are co-located at time t at a maximum distance of D, or in other words we want to compute the probability that they are close in a given time t. Let's refer to Figure 3.10 that represents the areas in which u and u' can be located at a given time t (as in Example 3.10). The equation

$$Pclose_D(t,u,u') = \iint_S p[loc_u(t) = l] \cdot p[loc_{u'}(t) = l'] \cdot c_D(l,l') \mathrm{d}l \mathrm{d}l'$$

is equal to

$$\int_{c_u} \int_{c_{u'}} p[loc_u(t) = l] \cdot p[loc_{u'}(t) = l'] \cdot c_D(l,l') \mathrm{d}l \mathrm{d}l'$$

since the probability outside of c_u for u is zero and analogously for u' in $c_{u'}$.

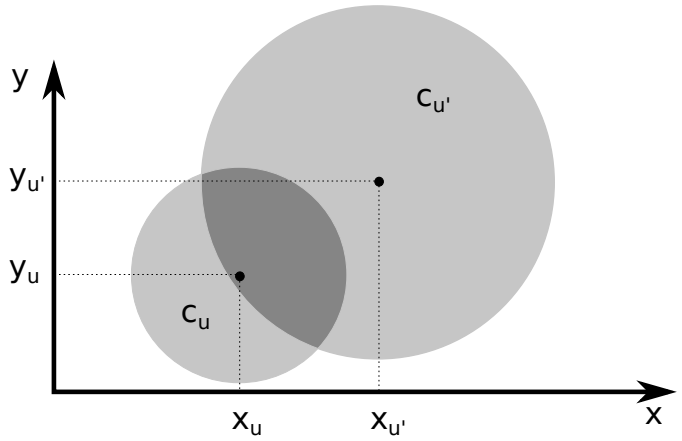


Figure 3.10: Approximate locations for users u and u' at time t

■ **Example 3.11** Let's suppose that the approximate location for u and u' are c_u and $c_{u'}$, respectively. If it not exists an intersection between c_u and $c_{u'}$, and furthermore the minimum distance between them is greater than the distance D, then the probability is null, since, in any possible locations they can be, they wouldn't be close. Let's now see how to compute the closeness probability when the intersection between c_u and $c_{u'}$ is not null, as in Figure 3.10. In the intersection i of c_u and $c_{u'}$ both the users have probability to be located, but, for each point $p \in i$ in which u can be located, we need to consider u''s probability for all the points within distance D from p. Let's assume that the computation of the double integral of Equation 3.2.6 is possible and have value $11/12$: if $\mathscr{K}$ is less than $11/12$ then a co-location threat is concrete. ■

Upper and lower bound for closeness probability

We want to find an upper and a lower bound for the closeness probability value, since it can be costly to compute even in the uniform probability scenario that we are considering.

- **Upper bound**

  Let's consider Figure 3.11: user u can be located in each point p of $iPlus_D$, i.e. the intersection between c_u and the circular region $cPlus_{u'}$ that is the circle with center in $(x_{u'}, y_{u'})$ and radius equal to r'_D. Considering p, the probability that u' is co-located can be upper bounded if we consider as possible all the points within distance D from $p \in iPlus_D$. Clearly this is not true, since as in Figure 3.11 we consider as not null the probability that u' is located in points as p_k. Hence:

$$Pclose_D(t,u,u') \leq \frac{A(c_u \cap cPlus_{u'})}{A(c_u)} \cdot \frac{A(c_{iD})}{A(c_{u'})} = \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}$$

  With this approximation if

$$\frac{A(iPlus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})} < \mathscr{K}$$

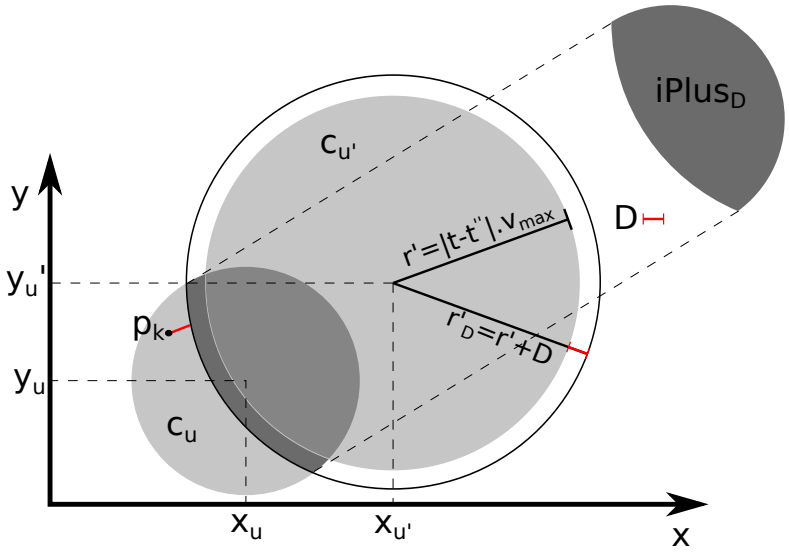


Figure 3.11: Computing the upper bound of $Pclose_D(t,u,u')$

then we can exclude a co-location threat, since, also

$$Pclose_D(t,u,u') \leq \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})} < \mathscr{K}$$

Formally:

Theorem 3.2.1 Let (x_u, y_u) be a known location at time t' for user u and let c_u, the circle centred in (x_u, y_u) with radius $r = (t - t')v_{max}$, be the approximated one at time t; let $(x_{u'}, y_{u'})$ be a known location at time t'' for user u' and let $c_{u'}$, the circle centred in $(x_{u'}, y_{u'})$ with radius $r' = (t - t'')v_{max}$, be the approximated one at time t. Furthermore, given a distance D, let $iPlus_D = c_u \cap \{\text{circle with center in } (x_{u'}, y_{u'}) \text{ and radius } r'_D = (r' + D)\}$. Hence, the closeness probability $Pclose_D(t,u,u')$ at time t for users u and u', in case of uniform probabilities, is upper bounded by:

$$Pclose_D(t,u,u') \leq \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}$$

where function $A(x)$ computes the area of the spatial region x.

Proof is reported in Appendix A.

Please note that the area of the region $iPlus_D$ can be computed analytically, and hence easily computed. The resulting formula, with $v_{max} = v$, is:

$$
\begin{aligned}
&A(iPlus_D)\\
&= [(t-t')v]^2 \arccos\left(\frac{2vt(vt''-vt'-D)-2t''Dv+v^2t'^2-v^2t''^2-D^2+d^2}{2dv(t-t')}\right)\\
&+ [(t-t'')v+D]^2 \arccos\left(\frac{2vt(vt'-vt''+D)-2vt''D+v^2t''^2-v^2t'^2+D^2+d^2}{2d[v(t-t'')+D]}\right)\\
&+ \sqrt{(2vt-vt''-vt'+D-d)(2vt-vt''-vt'+D+d)[(d+D)^2-(vt''-vt')^2]}
\end{aligned}
\tag{3.2}
$$

- **Lower bound**

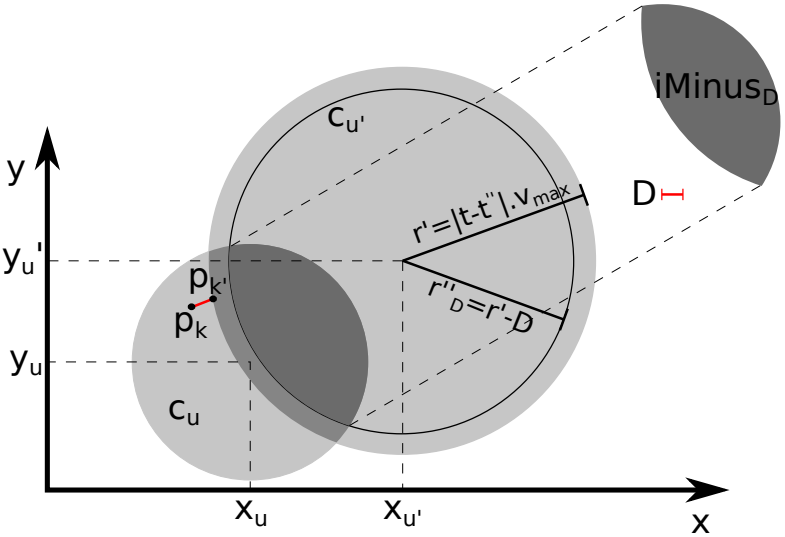


Figure 3.12: Computing the lower bound of $Pclose_D(t,u,u')$

Let's consider Figure 3.12: the user u can be located in each point p of the intersection $iMinus_D$ between c_u and the circular region $cMinus_{u'}$, that is the circle centred in $(x_{u'}, y_{u'})$ with radius equal to $r''_D = (t-t')v_{max} - D$. Considering all $p \in iMinus_D$, the probability that u and u' are co-located can be lower bounded if we consider all the points within distance D from p. Clearly in this way we "lose" some points in which u''s probability is not not null. For instance, referring to Figure 3.12, we can observe that both the probability that u is located in the point p_k and the probability that u' is located in $p_{k'}$ are not null: since the distance between p_k and $p_{k'}$ is equal to D, also the closeness probability in these two points is not null. However, this probability is not considered in the above formula. Hence:

$$
Pclose_D(t,u,u') \geq \frac{A(c_u \cap cMinus_{u'})}{A(c_u)} \cdot \frac{A(c_{iD})}{A(c_{u'})} = \frac{A(iMinus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}
$$

With this approximation if

$$\frac{A(iMinus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})} > \mathscr{K}$$

then we can be certain when a co-location threat occurs, since

$$Pclose_D(t,u,u') \geq \frac{A(iMinus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})} > \mathscr{K}$$

Formally:

Theorem 3.2.2 Let (x_u, y_u) be a known location at time t' for user u and let c_u, the circle centred in (x_u, y_u) with radius r, be the approximated one at time t; let $(x_{u'}, y_{u'})$ be a known location at time t'' for user u' and let $c_{u'}$, the circle centred in $(x_{u'}, y_{u'})$ with radius r', be the approximated one at time t. Furthermore, given a distance D, let $iMinus_D = c_u \cap \{\text{circle with centre in } (x_u, y_u) \text{ and radius=}(r' - D)\}$. Hence, the closeness probability $Pclose_D(t,u,u')$ at time t for users u and u', in case of uniform probabilities, is lower bounded by:

$$Pclose_D(t,u,u') \geq \frac{A(iMinus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}$$

where function $A(x)$ computes the area of the spatial region x.

Proof is reported in Appendix A.

Also the area of $iMinus_D$ can be computed analytically, analogously to the upper bound case.

Bivariate Normal distribution

Another possibility is modelling the users' probability f function through a normal distribution. We are now assuming that, given a know location $l = (x,y)$ for the user u at time t, his probability distribution is described by a bivariate normal distribution.

Definition 3.2.9 Let σ be a real value and v a constant speed. Given a user u, the probability that u is located in $l' = (x,y)$ at time t', given that u is located in $l = (\mu_x, \mu_y)$ at time t, is given by:

$$p[loc_u(t') = (x,y) | loc_u(t) = (\mu_x, \mu_y)] = \frac{1}{2\pi\sigma v \Delta t} \int_{-\infty}^{+\infty} e^{-(\frac{(x-\mu_{1x})^2 + (y-\mu_{1y})^2}{2\sigma v \Delta t})}$$

with $\Delta t = t' - t$.

The bivariate normal distribution, though not realistic, is able to express the concept that the quantity of information enclosed in a known location decreases as time goes by.

As in the general case, the the closeness probability in case of the bivariate normal distribution is the product of the probability of user u to be located in l at time t multiplied by probability of user u' to be in located in l' at the same time t, only if the distance between l and l' is not greater than D, as in Definition 3.2.6.

Discrete spatial domain

The value of the probability that a given user u is located in a given location l at time t can not be available for each point belonging to the spatial domain S, but only at higher granularity. For instance, it is reasonable that we do not precisely know the probability for the user to be on the second table of the pub, but only the overall probability that the user is located within the pub. Furthermore, since in many context the computation of the closeness probability (see Definition 3.2.6) can be very costly, a discretization of the spatial domain S can ease the computation.

Therefore, let's assume that the spatial domain S is subdivided in n equal regions $c_1, c_2, \cdots c_n$. We call such spatial regions *cells*.

We now want to compute the closeness probability $Pclose_D(t,u,u')$ in this discrete case. To compute this probability, we first need to define a piecewise function that takes two cells as input: if the distance[4] between the two cells is smaller or equal to the given threshold D, then the function value is equal to 1, while, in all other cases, its value is equal to 0.

Definition 3.2.10 Given a distance threshold D and two cells c and c', the *cell closeness* function $cReg_D(c,c')$ is equal to:

$$cReg_D(c,c') = \begin{cases} 1 & \text{if } d(c,c') \leq D \\ 0 & \text{otherwise} \end{cases}$$

where function $d(a,b)$ computes the distance between cells c and c'.

We can now compute the closeness probability in the discrete domain. In this case the double integral in Definition 3.2.6 is replaced by a double summation since the number of cells to compute is finite.

Definition 3.2.11 Given the spatial domain S subdivided in n equal cells $c_1, c_2, \cdots c_n$, given the users u and u', the probability $Pclose_D(t,u,u')$ that u and u' are close at a given time t is:

$$Pclose_D(t,u,u') = \sum_{i=1}^{n} p[loc_u(t) \in c_i] \cdot \sum_{j \in [1,n] | cReg_D(c_i,c_j)=1} p[loc_{u'}(t) \in c_j]$$

3.2.3 Group co-location

In the previous section we have discussed the pairwise co-location; now we want to present the group co-location. There are lots of similarities between the pairwise and the groups co-location definitions, and this is the reason why we will not get through all the details given in the previous pairwise section. There can be several motivation for a user for not wanting that a set of users is co-located with him, even if these users, separately, are not a problem for the user's privacy.

Beyond the practical reasons, we want to express what is a group co-location threat. To do that we first need to define a group co-location privacy preference. Differently to Definition 3.2.2, a group privacy preference for the user u do not contains a list of users, but a set of it. Each list represent the users with which the u does not want to be co-located at the same time. Formally:

[4] it can be the maximum, the minimum or the mean distance between the two cells, depending on the context.

Definition 3.2.12 A group privacy preference ϕ_u^G for the user u is a a set of tuples, where each tuple f_i is a set of two ore more identifiers of users with whom u does not want to be co-located at the same time.

■ **Example 3.12** Alice does not want to be co-located at the same time with Bob and Dave; also she is not willing to be co-located together with Elise, Francis and Georgia. Therefore her pairwise privacy preference will be:

$$\phi_{Alice}^G = \{\{Bob, Dave\}, \{Elise, Francis, Georgia\}\}$$

■

Please note that users contained in each list are not necessarily included in user's u pairwise privacy preference, since we consider the two privacy preferences as independent.

As in the pairwise co-location, we can have an explicit or implicit group co-location threat. Let's analyse both in the followings.

Explicit

Analogously to the pairwise co-location threat, we talk about an explicit one when only a resource is needed to infer the co-location. To detect a group co-location threat, considering the user's u privacy preference ϕ_u^G and (at least) one of its tuples f_i, it should exist a resource r in which all the users belonging to f_i are co-located with u. Formally:

Definition 3.2.13 An explicit group co-location threat for the user u, with group privacy preference ϕ_u^G, is the publication of a resource r such that:

$$\exists f_i \in \phi_u^G \text{ s. t. } \forall u_i \in f_i, (u_i \cap r.Udata) \neq \emptyset$$

■ **Example 3.13** Let's assume that Alice is using a Geo-Social Network and her group privacy preference are $\phi_{Alice}^G = \{\{Bob, Dave\}, \{Elise, Francis, Georgia\}\}$ (as in Example 3.12). If her friend Carl publish a post in the same social network, writing "Having a drink @Pub - with Alice and Bob" then the publication of this post is not a group co-location threat: Bob belongs to ϕ_{Alice}^G but there is a threat only if also Dave is included in the post. Otherwise, if Francis write in the social network "Shopping in Milan - with Alice, Barbara, Elise and Georgia", there is a co-location threat for Alice since all Elise, Francis, Georgia are included in the resource. Please note that the presence of another user, in this case Barbara, is not important. ■

Implicit

Let's consider now the implicit group co-location threat. Also this threat is very similar to the pairwise case. Here the threat for the user u happens when it exists a set of users (corresponding to a tuple in ϕ_u^G) that can be co-located with u, through two or more resources. Even in this case in the implicit co-location we loosen the concept of being co-located in the one of "being close", considering the distance D, as we have already discussed in the pairwise section.

We first need to define a piecewise function that takes as input a location l and a set $\mathscr{L}$ of other locations: if the distance between l and each of the location $l_i \in \mathscr{L}$ is smaller or equal than the given threshold D then the function value is equal to 1, otherwise, its value is equal to 0.

Definition 3.2.14 Given a distance threshold D, a location l and a set of locations $\mathscr{L} = \{l_1, \cdots l_n\}$, the *group closeness* function $c_D^G(l, \mathscr{L})$ is equal to:

$$c_D(l, \mathscr{L}) = \begin{cases} 1 & \text{if } \forall l_i \in \mathscr{L} d(l, l_i) \leq D \\ 0 & \text{otherwise} \end{cases}$$

where function $d(a,b)$ computes the (Euclidean) distance between locations a and b.

We now want to express the probability that given the user u and a set of users $\mathscr{U}$, all the users $u_i \in \mathscr{U}$ are close to u, always considering a fixed distance D.

Definition 3.2.15 Given a user u and a set of users $\mathscr{U} = \{u_1, \cdots u_n\}$, the $Pclose_D(t, u, \mathscr{U})$ probability that u's location is close to all the locations of users $\mathscr{U}$ at a given time t is:

$$Pclose_D(t, u, \mathscr{U}) = p[c_D^G(loc_u(t), \mathscr{L}) = 1]$$

with $\mathscr{L} = \{loc_{u_1}(t), loc_{u_2}(t), \cdots, loc_{u_n}(t)\}$.

We are now ready to define an implicit group co-location threat for the user u: if the probability $Pclose_D(t, u, \mathscr{U})$ between all the users $\mathscr{U}$ of a tuple in ϕ_u^G and u is greater that a given threshold at a time t then we are in presence of a co-location threat. Formally:

Definition 3.2.16 Given a real number $\mathscr{K} \in [0, 1]$ and a distance value D, an implicit group co-location threat for the user u, considering ϕ_u, is the publication of a set of resource $r_1, \cdots r_n$ such that:

1. $\forall i, j = 1, \cdots n, r_i \neq r_j$
2. $\exists f_i \in \phi_u^G$ s. t. $\forall u_i \in f_i, \exists j \in [1, n](u_i \cap r_j.Udata) \neq \emptyset$
3. $\exists t \in T,$ s.t. $Pclose_D(t, u, \mathscr{U}) \geq \mathscr{K}$

3.2.4 Summary

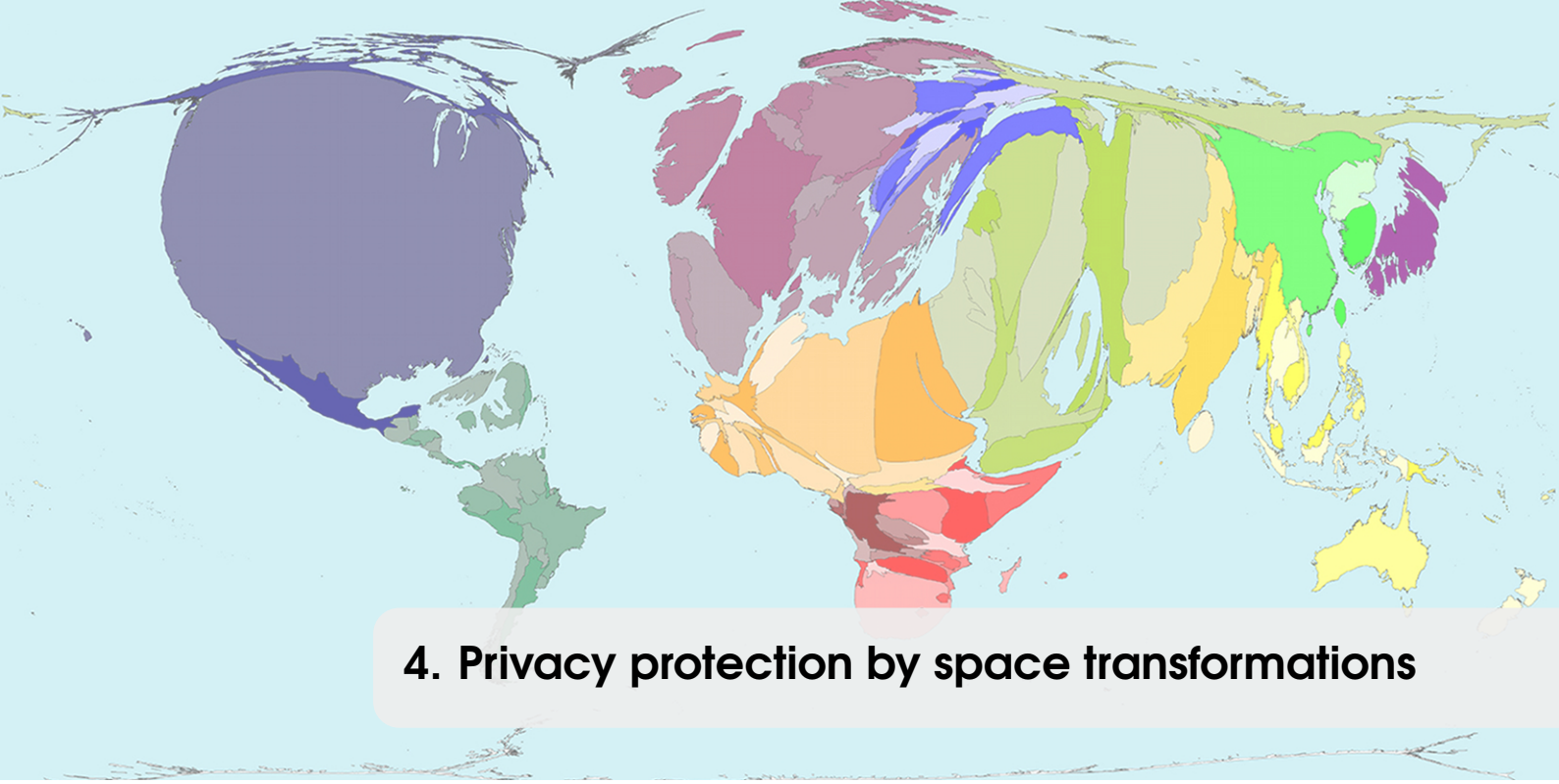
In this section we have analysed the co-location privacy issue. We have classified it with two orthogonal definitions, pairwise/group and explicit/implicit co-location threats. For each category we have formalised the conditions under which a co-location threat can occur. To capture the implicit co-location threat we have proposed a probabilistic definition whose computation can be costly. For this reason, we have presented three simple scenarios (uniform distribution, bivariate normal distribution and discrete spatial domain) in which the computation is more straightforward. These scenarios can help us to better understand the implicit co-location threat identification problem.

There are two major future works:

- Proposal of privacy preserving technique capable of counteract the co-location threat. Some of the existing privacy preserving techniques can be adapted to protect co-location privacy preferences, as well as novel privacy protection techniques can be specifically proposed for social networks, in which the amount of the data shared can make privacy protection a real

challenge.

- Perform a comprehensive experimental evaluation, to empirically evaluate if the co-location threat definitions proposed in this section are effective and capture the real user privacy threat. The experimental evaluation goals are to both verify the correctness of the formal framework and test the privacy protection effectiveness of the possible privacy preserving techniques.



4. Privacy protection by space transformations

Distance preserving transformations are among the techniques that may be used to preserve privacy while providing distance-based services involving moving objects (e.g., proximity services) and we have analysed them in Section 3.1.

As we have already discussed, it has been shown in the context of privacy preserving data mining, that distance preserving transformations can be subject to privacy attacks in presence of prior knowledge of the adversary [34]. Furthermore in Chapter 3 we have seen that this kind of attacks are feasible and harmful for the users' privacy even in a practical scenario.

In this chapter we will present a technique capable of computing the proximity between users, in which the users' privacy is obtained through locations' obfuscation, hence protecting their location privacy. Our contribution does not provide a technique to contrast the attacks mentioned before while preserving exact distance, but it effectively contrast attacks, as the one showed in Section 3.1, based on the most common prior knowledge that an attacker can have (knowledge of the transformation and probability distribution of the users positions) at the price of a distance distortion that can still enable very useful services.

4.1 The overall idea

There are plenty of privacy preserving techniques that can be used in proximity service. Among others criteria, the choice can depend on the efficiency and scalability needed by the service. Using distance preserving transformations as privacy protection has costs comparable with the solutions that do not provide privacy protection but it guarantees privacy only under the assumption that the adversary has no a-priori knowledge. Conversely, some cryptographic techniques guarantee protection also when the adversary knows the a-priori distribution but at the cost of higher communication and computation costs.

Among the techniques in the literature, we choose to work on:

1. the Longitude [40] privacy protection protocol as model of privacy techniques based on distance preserving transformations

2. the *C-Hide&Hash* [44] protocol as model of cryptography based privacy preserving technique

The aim of the following chapter is to propose a new protocol based on the following idea: when a user wants to compute the proximity with respect to all of his buddies, we first filter some of the buddies that are not in proximity by using an extended version of the Longitude protocol. Then, we apply the *C-Hide&Hash* protocol among the remaining users to discover those that are actually in proximity.

The technical contribution is the extended version of the Longitude protocol, which, thanks to a space transformation (through which the users' distribution in the space is forced to be uniform), is guaranteed to be safe also in presence of an a-priory probabilistic users distribution. We will start by formally defining the needed properties of that space transformation (see Section 4.2) and by presenting three algorithms capable of computing it (see Section 4.3). The overall technique structure will be detailed in Section 4.4.

4.2 Deforming space to obtain uniformly distributed data

The idea behind the extension to the Longitude protocol is to apply a space transformation function before using a Longitude-like proximity computation.

We are interested in a 2-dimensional space transformation function that given a finite number of points non-uniformly distributed in the space, changes the position of these points in order to obtain a uniform distribution. In our case, the points represent users' position on a geographical map. This transformation should have the property that if two points are close to each other in the original space, then they are not too far in the transformed space. Let's suppose that given two users u and u', they are in proximity only if the distance between their location is smaller or equal than a fixed distance δ. Therefore, given such distance threshold value δ in the original space S, the space transformation enlarges it at most δ' for any pair of points in S. Assuming that such a space transformation exists, we can easily define an efficient filtering technique to exclude that two users are in proximity. Actually, we can use a Longitude-like computation on the uniform distributed space to compute the distance between two users; if, in the transformed space the users are further from each other more than δ', then it can be excluded that the user are closer than δ in the original space. For those points in the transformed space for which the distance is less than δ' we can not be sure that are in proximity. In this case we should apply the *C-Hide&Hash* protocol to know with certainty if the two points are at a distance greater, less or equal than δ.

We now want to formalise the properties that this transformation function should have.

4.2.1 The space transformation function

Let S be the 2-dimensional domain $S : [0, x_{max}] \times [0, y_{max}]$ and let $p = (x, y)$ be a generic point belonging to S. A probability distribution function (PDF) $g(p)$ is defined over S. In our proximity service scenario the PDF g represents the probability distribution of the density of users in S.

The distribution of the users in a real environment is clearly not uniform since, for instance the population density (and as assumed in 3.1 this applies also for the users) in cities is higher than in the country regions.

Definition 4.2.1 A function f from S to S is called a *Space Transformation Function (STF) for g and S* if

1. it maps any continuos subspace of S to a finite number of continuous subspaces,
2. for all continuous subspace A in S, the following holds:

$$\int_{p\in A} g(p)\,dp = |A'|/|S|, \tag{4.1}$$

where $A' = \{f(p)|p \in A\}$ and $|A'|$ and $|S|$ denote the areas of A' and S, respectively.

Although the codomain of f is the same as the original space, in the following, when we need to distinguish between the original space and the transformed one, we denote them S and S', respectively.

Intuitively, the above definition says that an STF will map a subspace A in S to a subspace A' in S' such that proportion of the area of A' to that of S' is exactly the total volume of g on A (i.e., the cumulative probability). In this way, through the function f, we "smooth out" the density distribution g on S to a uniform distribution on S'. We denote this uniform distribution on S' as h, i.e., $h(p') = 1/|S'|$ for all $p' \in S'$.

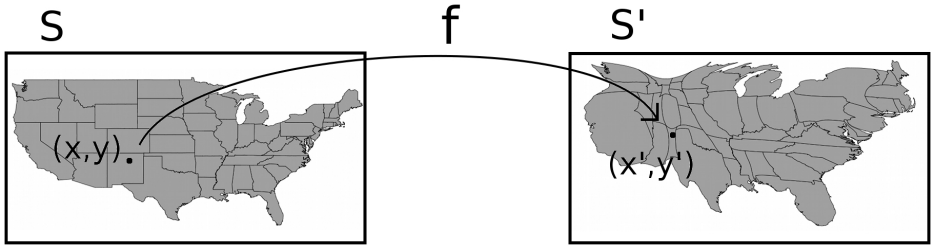
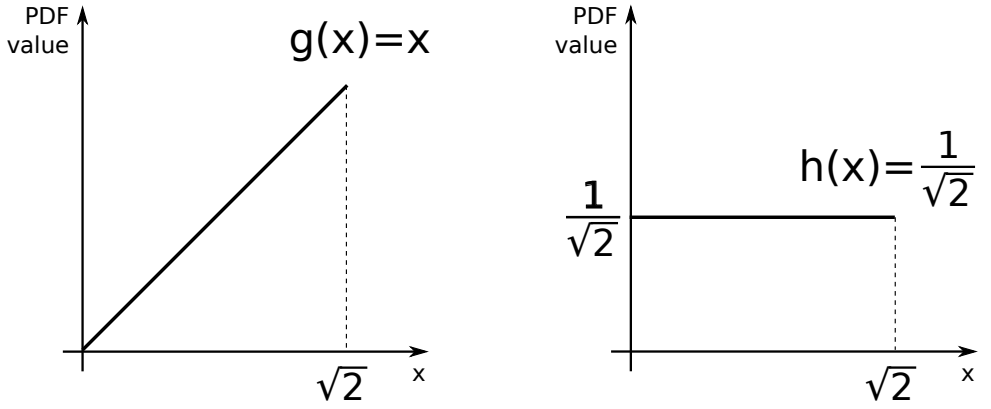


Figure 4.1: From S with non uniform distribution $g(p)$ to S' with uniform distribution $h(p')$

■ **Example 4.1** Let's consider an example in the one-dimensional case. Let g(x)=x be the PDF defined over S (see Figure 4.2 on the left). Note that $S = [0, \sqrt{2})$ and $\int_{x\in S} g(x)\,dx = 1$ as imposed by the definition of PDF. This PDF is obviously not uniform since in $x = 0$ the value of the PDF is 0, while in $x = \sqrt{2}$ the PDF value is $\sqrt{2}$.

The uniform distribution on S' is $h(x) = \frac{1}{\sqrt{2}}$.

Let's see how we can find an STF. Let each point $b \in S$ be mapped by $f(b)$ to a point $b' \in S'$ such that the area covered by $h(x)$ from zero to b' is equal to the area covered by $g(x)$ from zero to b. By definition of $g(x)$ and $h(x)$ we have that:



(a) g(x) (b) h(x)

Figure 4.2: The pdf $g(x) = x$ in S, and the corresponding pdf $h(x) = \frac{\sqrt{2}}{2}$ in S'

$$\int_0^b x dx = \int_0^{b'} \frac{1}{\sqrt{2}} dx$$

It follows that $b' = \frac{b^2}{\sqrt{2}}$. Hence, we can find the dependence of b' from b, that allow us to compute $f(x) = \frac{x^2}{\sqrt{2}}$. We claim that this function f is indeed an STF. Indeed, given a continuous subspace A in S, A must be a subinterval of $[0, \sqrt{2})$. Let this subinterval be $[a,b)$ (note that open or close on either side of A doesn't have any impact on the following discussion), and the interval $f(A)$ in S' is exactly $[f(a), f(b))$. The left-hand-side of Equation 4.1 gives $\int_a^b g(x)dx = \int_a^b x dx = (b^2 - a^2)/2$. The right-hand-side of the same equation gives $(f(b) - f(a))/|S'| = (b^2/\sqrt{2} - a^2/\sqrt{2})/\sqrt{(2)} = (b^2 - a^2)/2$. Hence, the equation holds and f is indeed an STF.

Let's make some numeric examples: if we consider two points of S', $x_a = 0.1$ and $x_b = 0.2$ we expect that they become closer in S' because they they belong to a region with low density. As expected, we have: $f(x_a) = 0.007$ and $f(x_b) = 0.02$. That is, a distance of 0.1 in S is mapped to a distance of 0.013 in S'. Conversely, if we consider other two points of S, $x_c = \sqrt{2} - 0.2$ and $x_d = \sqrt{2} - 0.1$, where the density ($g(x)$) is higher, we expect the distance in S' between them to be larger than in S. Indeed: $f(x_c) = 1.04$ and $f(x_d) = 1.22$. In other words, a distance of 0.1 is S is mapped to a distance of 0.18 in S'. ∎

Let $d = (p,q)$ be the euclidean distance of two points p and q in S. The STF transforms point p in $p' = f(p)$ and point q in $q' = f(q)$, and consequently their distance in the transformed space S' is $d' = d(p',q')$. For each possible pairs of points whose distance is smaller than or equal to d in S, the distance in the transformed space S' of the transformed points can assume several values. This clearly depends on the PDF in S: if the two points are located in an area in which the users' density is low then the distance will be enlarged in S, or shrunken if the points are in a high density region. Hence, for each pair of points of S whose distance is smaller than or equal to d, we denote with $minDistortion(d)$ and $maxDistortion(d)$ the minimum and maximum values, respectively, that d' can assume in S' from points with distance $\leq d$ in S.

Definition 4.2.2 Given a space transformation function $f : S \to S'$, two points $p, q \in S$ such that $p \neq q$, and a distance value d, the maximum distortion value $maxDistortion_f(d)$ is:

$$max_{\forall p,q \in S \text{ s. t. } d(p,q) \leq d} d(f(p), f(q))$$

Definition 4.2.3 Given a space transformation function $f : S \to S'$, two points $p, q \in S$ such that $p \neq q$, and a distance value d, the minimum distortion value $minDistortion_f(d)$ is:

$$min_{\forall p,q \in S \text{ s. t. } d(p,q) \leq d} d(f(p), f(q))$$

We are particularly interested in space transformation functions f such that the range $[minDistortion_f(d), maxDistortion_f(d)]$ is as small as possible. Intuitively, a small range provides a good approximation of d in S given a distance d' in S'. Hence we can consider this range as an indicator of the distance distortion associated with the function f.

In particular, given a distance d in S, if we can compute the $maxDistortion_f(d)$ value then, given two points $p' = f(p)$ and $q' = f(q)$ in S' whose distance is $\leq d'$, it is possible to make some inferences on the distance between the corresponding points p and q in S. Precisely, if the distance d' between the corresponding points in S' is more than $maxDistortion_f(d)$, then the two points in S are not closer than d. Formally:

Theorem 4.2.1 Let d be a distance, f a space transformation function and $maxDistortion_f(d)$ the maximum distortion value. Then, given two points $p' = f(p), q' = f(q) \in S'$ with $p, q \in S$ such that $d(p', q') > maxDistortion_f(d)$, then $d(p, q) > d$.

Clearly, if two points are close to each other in S', then, by knowing the maximum distortion value $maxDistortion_f(d)$, we cannot derive any information about their distance in S. Also, observe that the effectiveness of this filtering depends on the maximum distortion value of the STF f. For example, if d is small while $maxDistortion_f(d)$ is large compared to d, then the filtering is limited to the points that are further than d' in S'.

Based on this reasoning, we want to define a function f such that:

- $h(x, y)$ is a uniform distribution function, and
- $maxDistortion_f(d)$ is minimised.

4.3 Algorithm for space transformation

In this section we provide a set of algorithms capable of computing a STF f: however our algorithms do not guarantee that the $maxDistortion_f(d)$ is minimised, but only that the resulting users' distribution in S' is uniform. We are therefore considering a simplified problem with respect to the one in the previous section. In Section 4.5 we will experimentally evaluate the STF computed through the proposed algorithms to test their overall distortion and utility in a practical scenario.

The system model

We assume that the space S is subdivided, through a partition C, in a number of subregions, through a regular grid. Formally, given a space S over which a non-uniform probability density function of users is known and a partition C of S such that for each subregion c (element of C) the probability density function is known (assumed uniform), the space transformation function f, that we now want to define, maps each region $c \in C$ to a region c' obtaining a partition C' of S such that all the regions in C' have the same uniform density of users.

In Section 4.3.1 it is discussed the cartogram technique [16] and it is analysed its applicability for our problem. Then in Section 4.3.2 and in Section 4.3.3, the Recursive and the Greedy algorithms, respectively, are presented.

4.3.1 Cartograms

One of the existing techniques to obtain a plane S' with uniform distribution from a plane S where the distribution is non uniform is the use of cartograms.

A cartogram is a map produced using a cartographic technique where the mapped polygons (e.g., county or state boundaries) are stretched or shrunk based on the magnitude of the variable being mapped. Since our goal is to obtain a space in which the users are uniformly distributed, in our case this variable consists in the number of users in each subspace of S.

In literature there are several types of cartograms but the main are non contiguous cartograms, contiguous cartograms and Dorling cartograms, shown in Figure 4.3.

(a) Non-Contiguous (b) Contiguous (c) Dorling

Figure 4.3: Different Types of cartogram that represents the same data of population in California's counties. Source: http://www.ncgia.ucsb.edu/projects/Cartogram_Central

a) Non-contiguous cartogram. The geographical objects in this type of cartogram do not have to maintain connectivity with their adjacent objects. The connectivity is referred to as topography. Since the objects are free from their adjacent objects, they can maintain their shape. In a non-contiguous cartogram topology was sacrificed in order to preserve shape.

b) Contiguous cartogram. In this kind of cartograms topology is maintained (the objects remain connected with each other) but this causes a certain distortion in the objects shape. The objects must have the appropriate size to represent the attribute value, but they also have to maintain the shape of objects as best as possible. In order to have both characteristics, i.e. shape and contiguity, a contiguous cartogram introduces an approximation error.

c) Dorling cartogram. A Dorling cartogram maintains neither shape, topology nor object centroids. In this case the object are not enlarged or shrinked but are replaced with an object with a uniform shape of the appropriate size, usually a circle.

Clearly, cartograms a) and c) can not be used in our case, since we need to keep the distances in S' as much as possible close to the ones in S. Instead, we adopt the contiguous cartograms technique proposed by Gastner and Newman [16] as one of the techniques to uniform the user's distribution and we experimentally evaluate them in Section 4.5.

4.3.2 Recursive Algorithm

The idea of the algorithm is to recursively split the space in two regions each associated with a subset of the cells of the original partition. C of S and alter the dimensions of these two regions, such that the ratio between each region's area and the number of users within, is the same for both the regions. Indeed, the size of each region is chosen proportionally to the contained number of users (as given by the corresponding cells of C). The partitioning stops when a subarea has a single cell as the corresponding subset. Intuitively the subarea is the transformed space of that cell.

A matrix M corresponding to the cells partitioning S, with each element providing the number of users in that cell, is supposed to be globally defined (the main algorithm and all procedures have access to it). The algorithm (see Algorithm 3) is initially run with input S and an index range, given by two pairs of indexes of M denoting the lower left cell in M and the upper right cell in M. In the recursion steps the index range will denote the sub-matrix of M involved in that step.

The first step of the algorithm is the termination condition: if the index range denotes a single cell of M, then the variable S is the corresponding transformed space. Otherwise, the function divideAndResize() (see Algorithm 4) is invoked and returns two regions partitioning S each one with an associated index range. Analogously, the two index ranges are such that the denoted sub-matrices partition M. The algorithm is recursively called on each subregion, and index range.

The function *divideAndResize* is the core of the overall recursive algorithm. It takes as input a space S to be partitioned in two parts and an index range denoting the subset of cells of the matrix M containing user densities associated with the space S. The indexes are ordered by row and column with respect to the origin of S. The first step of the function determines if we partition horizontally or vertically, and the number of rows/columns of cells that should go in the first subarea.

In order to avoid recomputing cell by cell the sum of the users in M, we precompute partial sums. Given the matrix M where an element $u_{i,j}$ in $M[i,j]$ denotes the element in row i and column j, with $(0,0)$ indicating the index of the bottom left one, then the partial sum of users in C is defined as $P(i,j) = \sum_{m \leq i, l \leq j} u_{m,l}$. For example, in a $3 * 3$ grid $P(1,1)$ stores the sum of users

Algorithm 3 uniformRecursively

1: **Input:** Space $S = (x, y, l, h)$, an index range $R = (x_b, y_b, x_t, y_t)$. The matrix M containing the number of users u_{ij} for each cell $c_{i,j}$ of the grid $\mathscr{C}$ partitioning S is supposed to be globally accessible.
2: **Output:** a set of pairs $(S_{i,j}, (i, j))$ for each cell $c_{i,j}$ in R.
3: **Procedure:**
4: **if** $x_b = x_t$ and $y_b = y_t$ **then**
5: Result := $\{(S, (x_b, y_b)\}$
6: **return** Result
7: **else**
8: subArea[ ] := divideAndResize(S, R)
9: Result := uniformRecursively(subArea[0].S, subArea[0].R) $\cup$ uniformRecursively(subArea[1].S, subArea[1].R)
10: **return** Result
11: **end if**

in cells $c_{1,1}, c_{1,0}, c_{0,1}, c0, 0$. These partial sums will be precomputed, so that the computation of the number of users in a group of cells cam be done in constant time. Indeed, the number of users in a group of cells identified by the index range $R = (x_{min}, y_{min}, x_{max}, y_{max})$ is given by $CountUsers(R) = P(x_{max}, y_{max}) - P(x_{max}, y_{min} - 1) - P(x_{min} - 1, y_{max}) + P(x_{min} - 1, y_{min} - 1)$.

The decision of a vertical or horizontal division of the space as well as where the line dividing the space is positioned is taken by running Procedure 5. Intuitively, the procedure $findCut$ considers each possibility of vertical and horizontal division of the submatrix of M denoted by R: the procedure chose the parameters corresponding to the partition that minimises the difference in the number of users in the two corresponding subregions. A slightly different criteria is used in the alternative procedure 6 that, instead of minimizing the difference in terms of users, it minimizes the difference in terms of density (i.e., considering the number of cells in each group). The two procedures lead to different partitioning.

■ **Example 4.2** Let's consider a numeric example to better illustrate this technique. Let S be the original space that is a 27x27 unit square, divided into 9 cells (see Figure 4.4). The sum of all users is 27.

Initially, DivideAndResize is invoked with the whole S and with the whole range for the $9x9$ matrix.

Let's suppose that the $findCut$ algorithm's output is executed on range R, comprehending the whole space S and its output has "rows" as type, as in Figure 4.5(a). The result is the subdivision of the row in two subareas: the one in the upper part of Figure containing $5+3+1+3+3+3 = 18$ users, while the lower subarea containing $6+1+2 = 9$ users. The size of the subareas are resized according to the number of users (see Algorithm 4).

Then the algorithm considers the lower subarea (highlighted in red in Figure 4.5(b)). In this case, it is possible to divide only along the columns. Using a heuristic, we decide to divide along the left row. This forms two subareas: the left one containing 6 users, the right one containing $1+2 = 3$ users. The subareas are resized as in Figure 4.5(c). The subarea on the left is no

Algorithm 4 divideAndResize

Input: Space $S = (x, y, l, h)$, index range $R = (x_{min}, y_{min}, x_{max}, y_{max})$ denoting the portion of M associated with the space to be divided. These indexes allow to access elements $u_{i,j}$ of M.
Output: The pairs (S_1, R_1), (S_2, R_2) denoting the two divided geographical regions each one associated with the corresponding submatrix of M. R_i contains the indexes of lower and upper cells $(x_{il}, y_{il}, x_{iu}, y_{iu})$ of the submatrix.
Procedure:
$type, where \leftarrow findCut(M, x_{min}, y_{min}, x_{max}, y_{max})$ /* choose to partition by row or by column and how many rows/columns*/
$numUsers = CountUsers(R)$
if $type = row$ **then**
  $l_{c1} = l$; $l_{c2} = l$; $x_{c1} = x$; $x_{c2} = x$
  $u_{c1} = CountUsers(x_{min}, y_{min}, where, y_{max})$
  $u_{c2} = numUsers - u_{c1}$
  $h_{c1} = \frac{u_{c1}}{numUsers} \cdot h$; $h_{c2} = h - h_{c1}$
  $y_{c1} = y$; $y_{c2} = y + h_{c1}$
  $x_{1l} = x_{min}$; $y_{1l} = y_{min}$; $x_{1u} = where$; $y_{1u} = y_{max}$
  $x_{2l} = where + 1$; $y_{2l} = y_{min}$; $x_{2u} = x_{max}$; $y_{2u} = y_{max}$
else
  $h_{c1} = h$; $h_{c2} = h$; $y_{c1} = y$; $y_{c2} = y$
  $u_{c1} = CountUsers(x_{min}, y_{min}, x_{max}, where)$
  $u_{c2} = numUsers - u_{c1}$
  $l_{c1} = \frac{u_{c1}}{numUsers} \cdot l$; $l_{c2} = l - l_{c1}$
  $x_{c1} = x$; $x_{c2} = x + l_{c1}$
  $x_{1l} = x_{min}$; $y_{1l} = y_{min}$; $x_{1u} = x_{max}$; $y_{1u} = where$
  $x_{2l} = x_{min}$; $y_{2l} = where + 1$; $x_{2u} = x_{max}$; $y_{2u} = y_{max}$
end if
return $((x_{c1}, y_{c1}, l_{c1}, h_{c1}), (x_{1l}, y_{1l}, x_{1u}, y_{1u}))$ and $((x_{c2}, y_{c2}, l_{c2}, h_{c2}), (x_{2l}, y_{2l}, x_{2u}, y_{2u}))$.

5	3	1
3	3	3
6	1	2

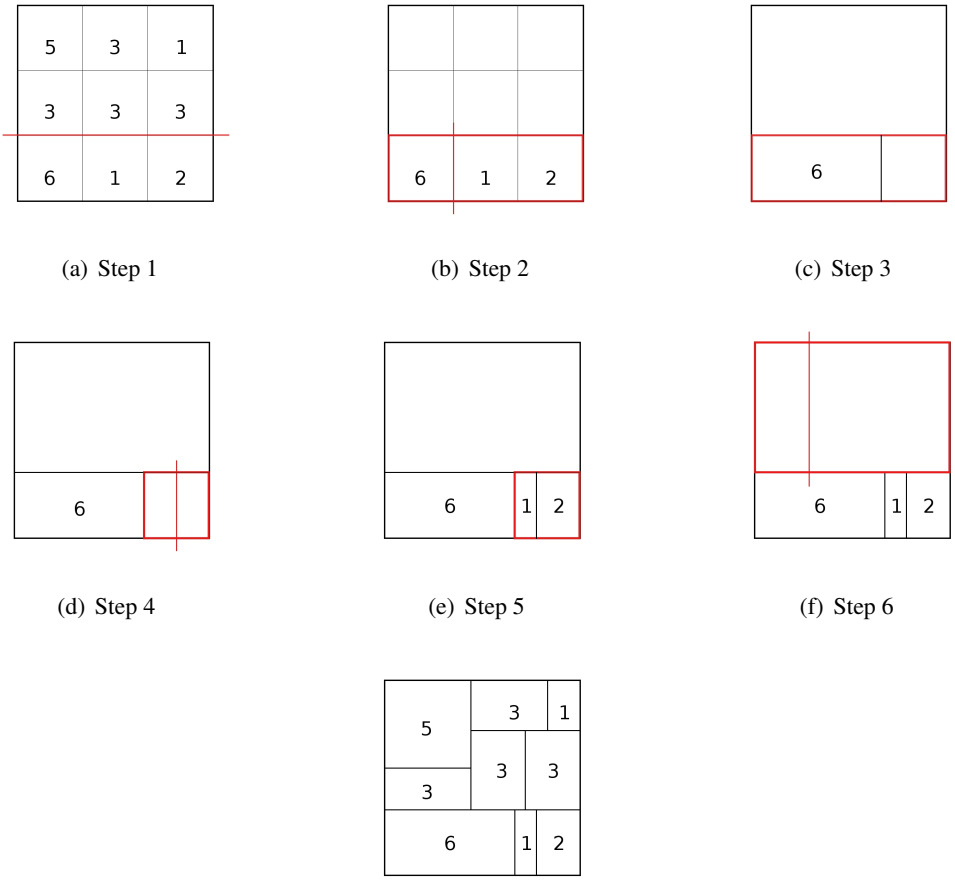
Figure 4.4: Recursive technique applied to the original space

Algorithm 5 findCut

Input: the index range $R = (x_{min}, y_{min}, x_{max}, y_{max})$.
Output: The pair of values $type$ and $where$ with $type$ = 'row' or 'column' and 'where' an integer.
Procedure:
$numUser \leftarrow CountUsers(x_{min}, y_{min}, x_{max}, y_{max})$
$mindif \leftarrow NumUser$ /* initialized to total number of users in M */
$type \leftarrow row$; $where \leftarrow y_{max}$ /* initialized to c1 covering all */
for $cut = x_{min} \rightarrow (x_{max} - 1)$ **do**
  $numUser1 \leftarrow CountUsers(x_{min}, y_{min}, cut, y_{max})$
  $numUser2 \leftarrow numUser - numUser1$
  if $|numUser2 - numUser1| < mindif$ **then**
    $mindif \leftarrow |numUser2 - NumUser1|$
    $type \leftarrow row$; $where \leftarrow cut$
  end if
end for
for $cut = y_{min} \rightarrow (y_{max} - 1)$ **do**
  $numUser1 \leftarrow CountUsers(x_{min}, y_{min}, x_{max}, cut)$
  $numUser2 \leftarrow numUser - numUser1$
  if $|numUser2 - numUser1| < mindif$ **then**
    $mindif \leftarrow |numUser2 - NumUser1|$
    $type \leftarrow col$; $where \leftarrow cut$
  end if
end for
return $type$ and $where$.

Algorithm 6 findCut-S

Input: the index range $R = (x_{min}, y_{min}, x_{max}, y_{max})$.
Output: The pair of values $type$ and $where$ with $type$ = 'row' or 'column' and 'where' an integer.
Procedure:
$numUser \leftarrow CountUsers(x_{min}, y_{min}, x_{max}, y_{max})$
$numCells \leftarrow (x_{max} - x_{min} + 1) * (y_{max} - y_{min} + 1)$
$mindif \leftarrow NumUser/numCells$ /* initialized to total density in the area */
$type \leftarrow row$; $where \leftarrow y_{max}$ /* initialized to c1 covering all */
for $cut = x_{min} \rightarrow (x_{max} - 1)$ **do**
  $numUser1 \leftarrow CountUsers(x_{min}, y_{min}, cut, y_{max})$
  $numUser2 \leftarrow numUser - numUser1$
  $numCells1 = (y_{max} - y_{min} + 1) * (cut - x_{min} + 1)$
  $numCells2 = numCells - numCells1$
  if $|numUser2/numCells2 - NumUser1/numCells1| < mindif$ **then**
    $mindif \leftarrow |numUser2/numCells2 - NumUser1/numCells1|$
    $type \leftarrow row$; $where \leftarrow cut$
  end if
end for
for $cut = y_{min} \rightarrow (y_{max} - 1)$ **do**
  $numUser1 \leftarrow CountUsers(x_{min}, y_{min}, x_{max}, cut)$
  $numUser2 \leftarrow numUser - numUser1$
  $numCells1 = (cut - y_{min} + 1) * (x_{max} - x_{min} + 1)$
  $numCells2 = numCells - numCells1$
  if $|numUser2/numCells2 - NumUser1/numCells1| < mindif$ **then**
    $mindif \leftarrow |numUser2/numCells2 - NumUser1/numCells1|$
    $type \leftarrow col$; $where \leftarrow cut$
  end if
end for
return $type$ and $where$.



(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5 (f) Step 6

(g) Final subdivision

Figure 4.5: An example of execution of the Algorithm *uniformRecursively*

longer divisible, since is composed only by one cell. Conversely in the sub area on the right the algorithm divide again along the columns, as in Figure 4.5(d). The two subareas are then resized: the one on the left containing 1 user has size that is half the one on the right, that contains 2 users (see Figure 4.5(e)).

Since the two subareas in the last step are composed by only one cell, we ended the run of the algorithm regarding the sub-bottom we had formed in the first step, and we can now consider the upper subarea 4.5(f)). The algorithm continues to divide the space, with the same criteria, until each subregion created by a split is composed by a single cell as in Figure 4.5(g).

■

Theorem 4.3.1 The space transformation obtained by Algorithm 3 has the property defined in Section 4.2.1 and the resulting space S' has a uniform density probability distribution.

Theorem 4.3.2 Algorithm 3 has a worst case computational complexity linear in terms of the number of uniform density cells partitioning S.

Proofs of the theorems are reported in Appendix A.

4.3.3 Greedy Algorithm

The algorithm presented in the previous section modifies the cells' sizes according to their users' density. In this section we want to change the perspective and adopt the dual strategy: the aim is to uniform the users' distribution by "moving" the users from cell to cell and keeping the cells' shapes and areas unaltered.

Similarly to the Recursive Algorithm, the Greedy Algorithm that we are going to present in this section assumes that S is discretised into cells each one having, internally, a uniform distribution. However, in this case, the idea of the algorithm is not to change the size of the resulting rectangles. Instead, we "move" the users among cells until a uniform distribution is achieved.

The algorithm is composed by two functions: a preprocessing function, that we call *DensityLeveling* and the on-line transformation function *PointsMapping* that, given the results of the *DensityLeveling* function, computes, for each point in S, the corresponding point in S'.

The DensityLeveling algorithm

The *DensityLeveling* is a function whose aim is to "move" the users among cells in such a way that in the final subdivision all cells have the same number of users. These "movements" of users are represented by pointers between cells. The *DensityLeveling* function guarantees that the total number of pointers that point to each cell i is constant and is equal to the average number of users in the cells (i.e., the total number of users divided by the total number of cells, we assume this value to be an integer).

The *DensityLeveling* algorithm computes the pointers among cells, such that to each cell it is associated the same number of users.

The pseudocode of the *DensityLeveling* algorithm is presented in Algorithm 7. The input of the algorithm is the set of cells $\mathscr{C}$, and for each $c \in \mathscr{C}$ the number of users u_c within.

The algorithm is iterative and greedy. At each iteration of this algorithm, some users are "moved" from the cell with the highest density (the *source* cell) to the closer cell having a number of users below average (the *target* cell). The general idea is that cells with higher number of users should give some of their users to cells with lower (under mean) number of users.

More precisely, at each iteration the source cell is the one with the highest number of users (see Line refwhileCond of Algorithm 7). In case more than one cells have the highest number of users, one of these cells is chosen randomly or by using a heuristic. The target cell is the one having a number of users below average that is closer (according to the Manhattan distance) the the source cell. Again, if this cell is not unique, the target cell is chosen randomly or using an heuristic. The source cell gives to the target cell the users that are required by the target cell to reach the average number of users (i.e., the total number of users divided by the total number of cells). However, the source cell gives at most the number of users that it has above the average. Formally, given i the source cell, j the target cell and AVG the average number of users in each cell, the number of users that i gives to j is given by: $min(|i| - AVG, AVG - |j|)$, computed in

Algorithm 7 DensityLeveling

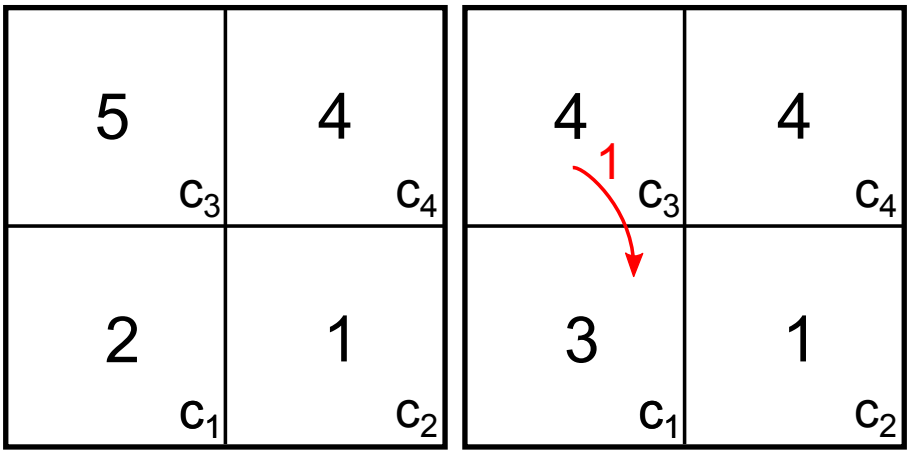
Input: Space S, a set $\mathscr{C}$ of cells c, the number of users u_c for each cell $c \in \mathscr{C}$
Output: the set of cells $\mathscr{C}$
Method:
$U = \sum_{c \in \mathscr{C}} u_c$
$\bar{v} = \frac{U}{|\mathscr{C}|}$ //$\bar{v}$ is the mean number of users
for all cells $c \in \mathscr{C}$ **do**
    divide c in u_c areas A_c
    compute the set of pointers $P_c = \{a_1 : c, a_2 : c, \cdots a_{u_c} : c\}$ where $a_i \in A_c$
end for
while $\exists \bar{c}$ such that $\forall c \in \mathscr{C}\, u_{\bar{c}} \geq u_c$ and $u_{\bar{c}} > \bar{v}$ **do**
    $source = \bar{c}$
    $target$= cell $\hat{c}$ closest to $\bar{c}$ whose $u_{\hat{c}} < \bar{v}$
    $toMove = min(u_{source} - \bar{v}, \bar{v} - u_{target})$
    $(\mathscr{L}, \mathscr{K}) = globalMove(toMove, source, target, P_{source}, P_{target})$
    $updatePointers(\mathscr{C}, \mathscr{K})$
    $u_{source} = u_{source} - toMove$
    $u_{target} = u_{target} + toMove$
end while
return the set of cells $\mathscr{C}$

Procedure 8 globalMove$(n, source, target, P_{source}, P_{target})$

Input: an integer n, a cell *source*, a cell *target* and a set of pointers P_{source} of the cell *source*, a set of pointers P_{target} of the cell *target*
Output: the pair ($\mathscr{L}$, $\mathscr{K}$) where $\mathscr{L}$ is a set of cells and $\mathscr{K} = P_{c_1}, \cdots P_{c_{|\mathscr{L}|}}$ is a set where each P_{c_i} is the set of pointers of the cell $c_i \in \mathscr{L}$
Method:
$\mathscr{L}$= getPath(*source*,*target*)
$\mathscr{K} = \emptyset$
for all cells c in $\mathscr{L}$ **do**
    n pointers in P_{source} are updated: the corresponding areas $a_i \in P_{source}$ points to c
    $\mathscr{K} = \mathscr{K} \cup P_{source}$
    source=*c*;
end for
return $(\mathscr{L}, \mathscr{K})$

Line 13 of Algorithm 7, where $|i|$ is the number of users in i and $|j|$ is the number of users in j. The "users movement" is performed by the *globalMove* procedure, whose pseudocode is shown in Procedure 8. Then the pointers changed during the *globalMove* procedure are updated, as well as the number of users of the source and the target cells (see Lines 15-17 of Algorithm 7).

The algorithm terminates when all the cells have the same number of users, and the set of cells $\mathscr{C}$ together with the pointers for each cell $c \in \mathscr{C}$ is returned. Please note that at each iteration either the source or target cell (or both) reaches the average number of users. Also, no cell having the average number of users is used as source or target cell. Hence, at each iteration, the number of cells having a number of users different from the average decreases.



(a) Space S (b) Step 1

(c) Step 2 (d) Step 3

Figure 4.6: Users are moved from cell to cell by the *DensityLeveling* algorithm

■ **Example 4.3** The original space S is divided into 4 cells, as in Figure 4.6(a). The bigger numbers represents the number of users, while the c_1, c_2, c_3, c_4 are the cell's indexes. Since the

sum of all the users in the space S is 12,the average number $\bar{v}$ is equal to 3.

In iteration 1, the source cell, i.e. cell c_3 with 5 users, is considered as the one containing the highest number of users; the corresponding target cell, i.e. the closest cell having a number of users below average is cell c_1, that contains 2 users. The source cell can give a maximum of $5-\bar{v}=5-3=2$ users, while the target cell require only $\bar{v}-2=3-2=1$ user. Consequently, as in Figure 4.6(b), since $toMove = min(2,1) = 1$ (see Line 5 of Algorithm 7), one user is given from cell c_3 to cell c_1: therefore the number of users of the source cell is decreased by one and the target is increased by one. The pointers are modified, according to the *globalMove* procedure: through the *globalMove* procedure, one area $a_i \in A_{c_3}$, i.e. one of the 5 areas in which c_3 is subdivided, is now pointing to the cell c_1. The new number of users of cells c_3 and c_1 is computed in Lines 16-17.

In the second iteration of the *DensityLeveling* algorithm, there are 2 cell that are candidate source cells (cells c_3 and c_4), since both have 4 users: let's assume that the cell c_4 is chosen as source cell (randomly or through an heuristic). In this case the target cell is the one with only 1 user, i.e. cell c_2. The number of users that should move from cell c_4 to cell c_2 is $toMove = min(4-3,3-1) = min(1,2) = 1$. In Figure 4.6(c) we can see that one user moves from the source cell to the target cell, lowering the users of cell c_4, and increasing by one the users of cell c_2.

In the third and last iteration the source cell is again cell c_3 with 4 users and the last cell with a number of users below average is cell c_2 with 2 users. The source cell gives one user to the target cell (see Figure 4.6(d)).

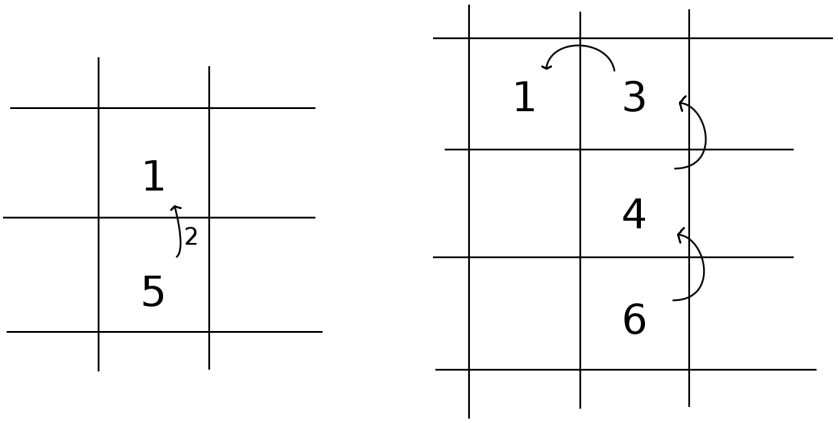
Since now the number of users is the same in each cell, the *DensityLeveling* algorithm terminates its execution. ∎

We now discuss how to "move" users from the source cell to the target one. If the two cell are adjacent (horizontally or vertically), then, the target cell could "receive" the user/users directly from the source cell. If the two cells are not adjacent, naively adding a pointer from the source to the target would result in a distortion directly linear to the distance between the source and the target cells. Since our aim is to keep the distortion of the space S' as small as possible, we need to find another solution. For this reason, the *globalMove* procedure moves users from the source to the target by choosing a path between the two cells and by "exchanging" users only between two adjacent cells.

The pseudocode of the *globalMove* procedure is shown in Procedure 8. The input of the procedure are an integer *toMove*, the source and the target cells, and the sets of pointers P_{source} and P_{target} for the source and the target cells, respectively. The procedure first computes the set of cells $\mathscr{L}$, i.e. the path of pairwise adjacent cells, that will be involved in the "users movement" (see Line 4 of Procedure 8). In each iteration, the procedure considers a pair of adjacent cells belonging to our path $\mathscr{L}$: the *source* cell and the target c. If the procedure has to move n users, then n subareas a_i belonging to the cell source are selected (Line 7 of Procedure 8): the cells associated to this subareas are changed such that they now point to the cell c. Then c becomes the new source cell and the one that follows it in the path $\mathscr{L}$ is considered as the new target cell c. The procedure ends when all the cells belonging to $\mathscr{L}$ are processed, and with them all the set of pointers P_c for all $c \in \mathscr{L}$, that the procedure stores in the variable $\mathscr{K}$ (Line 8 of Procedure 8). The set $\mathscr{L}$ and the set $\mathscr{K}$ are returned as output.

■ **Example 4.4** Consider Figure 4.7, and let's suppose that the average number of users per cell $\bar{v}$ is 3.

In Figure 4.7(a) is represented a source cell, the one with 5 users and a target cell, with one user only, that are contiguous: in this case, as described above, the movement of the users is performed directly by adding a pointer from the source to the target.



(a) Adjacent cells (b) Non-ajacent cells

Figure 4.7: User are moved from source cell and target cell

In Figure 4.7(b) the source cell, the one with 6 users, and the target one, with one user, are not close. Then a path is selected, so that only contiguous cells can "exchange" users through pointers. In this case the source cell gives part of its users to the upper cell, the upper cell yields the same number of users received from the target cell to the cell over it. Finally the cell containing 3 users can give to the target cell the users that (indirectly) came from the source cell.

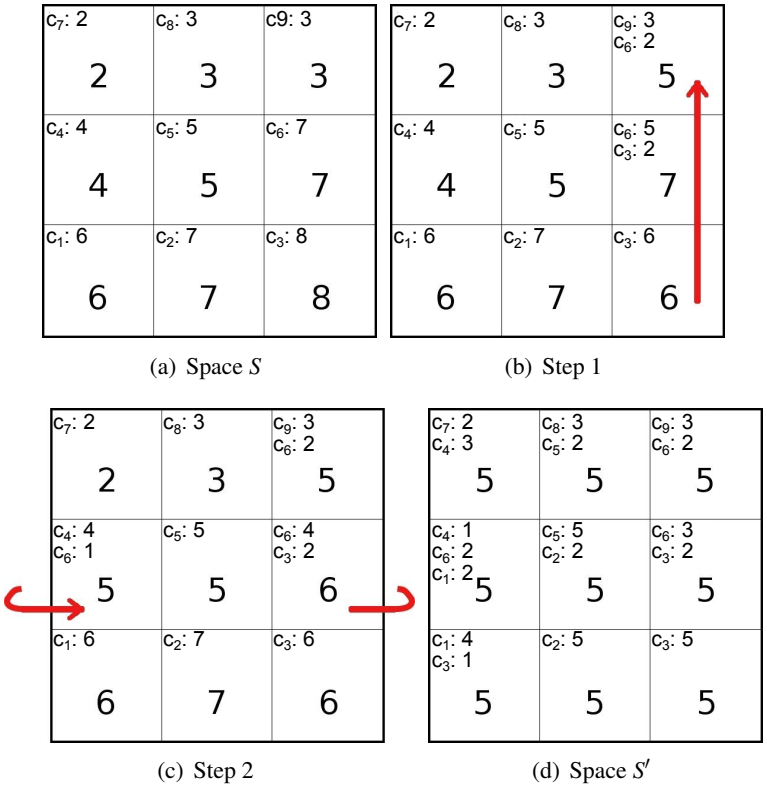
■

We'll not go into the details on how to choose a path from the source to the target cells: this will be computed through some heuristics aimed at minimizing the overall distortion of the space S'. Let's see now a more complete example.

■ **Example 4.5** The original space S is divided into 9 cells, as in Figure 4.8(a). The average number of users for cell $\bar{v}$, since all the users in the space are 45, is equal to 5: the purpose of the algorithm is to have 5 users in each cell.

Figure 4.8(a) represent the space S: the total number of users in each cell is reported in the centre. In the top left part of each cell there is a pair $c_i :_i$ where c_i is the cell id and a_i is the users' number in that cell. To be precise, the pair $c_i : a_i$ in cell c_j means that a number a_i of subareas belonging to c_i are now pointing to c_j: indeed, before running the *DensityLeveling* algorithm all the cells point to themselves with a number of subareas equal to the original number of users in the cell, as in Figure 4.8(a).

At the first iteration, the algorithm considers the cell having the highest number of users. In this case, this is the cell c_3 that has 8 users within. The closest cell (distances are measured in terms of manhattan distance) with a number of users below the average is cell c_9. So, the



(a) Space S (b) Step 1

(c) Step 2 (d) Space S'

Figure 4.8: Example of execution of the greedy algorithm

algorithm "moves" 2 users from cell c_3 to cell c_9. This is performed in two steps: first 2 users are moved from cell c_3 to cell c_6, then 2 users are moved from cell c_6 to cell c_9. Consider Figure 4.8(b): the pointers of cell c_9 denote that 3 users "comes from" cell c_9, while 2 users "comes from" cell c_6. Similarly for cell c_6.

In the next iteration (consider Figure 4.8(c)), there are two cells with a users number equal to 7. We chose (randomly or using an heuristic) to consider cell c_6. The closer cell with a number of users below the average is cell c_4, so a user is moved from cell c_6 to cell c_4. The pointers in each cells report this movement as shown in Figure.

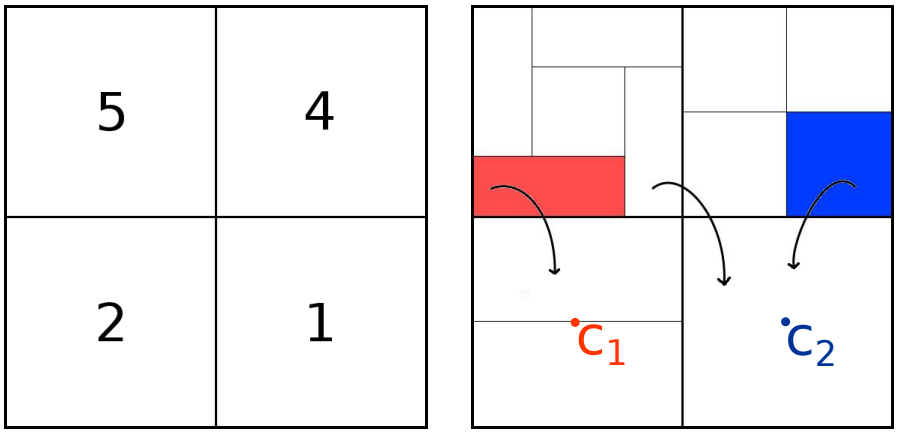
The algorithm continue to iterate, moving the users from cell to cell, until it is reached an uniform distribution, as the one in Figure 4.8(c). Consider, for example, cell c_3: the original number of users was 8. Now, the total number of users associated to this cell are 5; the "missing" 3 users have been moved to cells number c_1 and c_6 (1 unit in the former, 2 in the latter), as described by the pointers.

■

The *PointsMapping* function

Given the result of the *DensityLeveling* algorithm, the mapping of a point p in S to a point p' in S' is computed as follows: first, the cell c_i containing p is computed. Given c_i, it is computed

the subarea a containing p, as well as it is retrieved P_{c_i}, i.e. the pointers associated to the area a. Then, the centre of the cell pointed by the pointer is returned as the transformed location p'.



(a) Regular subdivision in S (b) Uniformed subdivision in S'

Figure 4.9: Before and after execution of the Greedy Algorithm

■ **Example 4.6** The original space S is divided into 4 cells as in Figure 4.9(a), in which the numbers within each represents the number of users within. The total number of users in S is 12, and consequently the average number of user for each cells $\bar{v}$ (i.e., the total number of users divided by the total number of cells,) is equal to 3. Each cell is divided in a number of subareas that is equal to the total number of user of the cell. For exmple in Figure 4.9(a) the cell with 5 users is divided in Figure 4.9(b) into 5 areas, the cell on its right containing 4 users is divided in 4 subareas, and so on. The output of *DensityLeveling* function is a set of pointers for each cell: every cell has associated a number of subareas that is equal to 3, that is the average number of users. In Figure 4.9(b) the pointers from a cell to another are represented through arrows. For simplicity, arrows that go from one cell to the same cell are omitted.

So, given the output of *DensityLeveling* function, we can map a point from S to S'. For instance a point p located in the red region of Figure 4.9(b) (the bottom left corner of the cell containing 5 users),following the corresponding pointer, will be transported in S' in the centre c_1 of the cell under it. Similarly, a point q locate in the blue region of Figure 4.9(b) (the bottom right of the cell with 4 users) will be mapped in S' in the centre c_2 of the bottom right cell. ■

Theorem 4.3.3 The space transformation performed by the *densityLeveling* algorithm has the property defined in Section 4.2.1 and the resulting space S' has a uniform density probability distribution.

With respect to the Recursive algorithm, with this algorithm we do not need to keep in memory all the cells' sizes and shapes, since they are the same for all the cells and the borders of each cell can be computed in constant time, given its position within the cells' matrix M. The weak point of this algorithm, due to the presence of numerous pointers, is that it could not be easily scalable. Depending on the area of the space S, and on the cells size, we can deal with

a number of cells of more than 10^9 elements: managing pointers between these cells is clearly computationally challenging.

4.4 Application to privacy preserving proximity notification

In this section we show how the use of a space transformation can be used to efficiently implement a privacy-preserving techniques resistant to attacks based on prior knowledge on the density of population in different areas. Indeed, if the users' distribution is known by the adversary and he knows the distances between some pair of users, he is able to infer users' locations quite precisely, as we have previously shown in Section 3.1.

The LBS service we are considering computes proximity among two moving users while preserving location privacy for both. Let's suppose we have a user A that wants to check if his buddy B is in proximity or not.

By using the IP address of a device A and public IP geocoding services, it is easy for a client on the device itself as well as for any server receiving requests from the device to obtain the space region S_A where the device is approximately located. Since the SP receives messages from the users whose proximity needs to be determined a trivial way to filter out users that are certainly not in proximity is to use the IP geocoding information. However, when IPs are released by 3G operators these regions are very imprecise and can easily cover hundreds of miles or an entire nation. We therefore need more accurate methods to understand if users that are reported by IP geocoding in the same or in an adjacent space region are actually in proximity.

The general idea is to apply the uniform density transformation separately to each of the space regions for which the SP cannot have more precise location information. As a simplifying example, suppose that from the IP address the SP cannot derive location information more precise than at the granularity of "state". Then, we apply the uniform density transformation within each state and compute for each one the value of the maximum distance distortion rate. With respect to applying the space transformation function to the whole world, by considering small regions it is more likely that a transformation with a lower maximum distance distortion can be found.

4.4.1 Proximity Computation algorithm

The algorithm can be summarised as follows:

1. Precompute for each space S_i the space transformation function as illustrated in Subsection 4.3.

2. A proximity request is sent by a user's device to the SP including a point p'' derived as follows:

   (a) the actual location p is transformed in p' by using the precomputed space transformation function for the space S_p identified by IP geocoding

   (b) According to the Longitude technique [40], p'' is derived by applying a solid transformation to p' by adding a secret value to each of its coordinates. In the underlying box the Longitude technique is briefly and intuitively described, please refer to [40] for all the technical details. The secret is shared only with participating friends.

Longitude

We recall that the idea of Longitude is to apply a modular translation on the coordinates of a user's location. The shift on the two axis is the secret that each user share with his buddies.
The main steps of the Longitude protocol are the following: each time A wants to check whether B is in proximity, A encrypts the cell c_A where A is located and sends it to the SP. The encryption is equivalent to project c_A in a toroidal space, and then apply a solid transformation. Upon receiving the request, the SP sends a message to B requiring a location update. B encrypts the cell c_B where B is located and sends the result to the SP. Note that B uses the same key as A, generated from their common secret. The encryption function is designed in such a way that the SP, upon receiving a request from A and an answer from B with cells encrypted with the same key, can compute the distance in the toroidal space, which we call modular distance. The SP compares the modular distance with the proximity threshold δ and sends the result as a boolean value to the requester A that computes whether B is in proximity or not. The encryption function is such that, if A sends his location cell to the SP using the same encryption key in different instants and while being in different cells, and the SP is aware of this, he can possibly learn some information about the movement of A, and hence about his location. For this reason, A changes the encryption key each time he communicates his location cell to the SP.

3. Location updates can be solicited by the SP based on a proximity request or spontaneously sent by the users, but in all cases they include a point q'' computed as illustrated above for p''; note that q may belong to a space S_q different from S_p.

4. When the SP has the point q'' received from a friend of the user, it compares the euclidean distance $d(p'', q'')$ with the value $\Omega pq * \delta$ where Ω_{pq} is the maximum among $maxDistortion_f(\delta)$ in the space S_p and $maxDistortion_f(\delta)$ in the space S_q and δ is the proximity threshold. If the distance is greater than that value, then the friends are certainly *not* in proximity.

5. If the users are not filtered out, then they can be in proximity: for compute the proximity with precision we run the more computationally expensive *C-Hide&Hash* protocol [44].

The C-Hide&Hash protocol

In order to ensure user's privacy, the *C-Hide&Hash* protocol adopts symmetric encryption techniques. We assume that each user A has a key K_A that is shared with all of his buddies and is kept secret to everybody else. Hence, each user A knows his own key K_A and one key K_B for each buddy B.

Periodically, once in each update interval u_i, each user A provides his location information to the SP, sending to the SP the hased index of the granule g_A where he is located. Since the service provider does not know the key K_A of A, his location can not be computed by the SP.
In the C-Hide&Hash protocol, when the user A issues a proximity request, he starts a two-party set inclusion protocol with the SP. The protocol is a secure computation, and consequently, the SP does not learn whether A is in proximity with his buddies,and A only learns, for each of his buddies B, whether B is in proximity or not, without learning in which granule B is located. The secure computation exploits a commutative encryption function C. In addition to the symmetric keys mentioned before, at each proximity request, the requesting user and the SP each generates a random key that is not shared with anyone else. We denote these keys K_1 for user A and K_2 for the SP. The proximity request sub-protocol is divided into three steps.

(a) A computes the set ES of indexes i such that the minimum distance between the location of A and the granule with index i is less than or equal to the proximity threshold δ. Each element of S is first hashed using the key K^{ui} , which is obtained as the ui-th value generated by the keystream initialized with K_B . In this case, ui is the index of the update interval preceding the current one. Then, the result is encrypted, using the with the C function using key K_1 that is randomly generated and it is sent to the SP.

(b) Upon receiving the set ES the SP encrypts each element of ES using the C function using key K_2 , which is randomly generated. The result is the set ES'. Then, it retrieves h_B that is the value of the index of the granule where B is located, hashed with the key K^{ui}, at the update interval ui . Since ui is the update interval preceding the current one, the location update policy assures that a location update with update interval ui has already been issued by every buddy B. Finally, the SP encrypts h_B with the C function using the key K_2 . The resulting value h', together with B and ES', is sent to A.

(c) A obtains h'' as the encryption of h' with the key K_1 and checks if the result is in ES' (let's recall that C function has the commutative property). If this is the case, then B is in proximity, otherwise he is not.

The correctness of the above algorithm is supported by the following reasoning.
The distance $d(p'',q'')$ is given by the sum of a distance component within S_p (from p'' to a point on the border of the S_p region) a (possibly empty) component outside both S_p and S_q (connecting the two points on the borders) and a component in S_q. Then, we have: $d(p'',q'') = d(p',q') \leq \Omega_{pq} * (d(p,q) - mindist(S_p,S_q)) + mindist(S_p,S_q)$, where $mindist()$ is the minimum distance between any pair of points on the border of S_p and S_q. Note that when the two spaces are adjacent, $mindist(S_p,S_q) = 0$ and the above inequation becomes $d(p'',q'') = d(p',q') \leq \Omega_{pq} * d(p,q)$. Moreover, $\Omega_{pq} * (d(p,q) - mindist(S_p,S_q)) + mindist(S_p,S_q) \leq \Omega_{pq} * d(p,q)$.

Hence, independently from the fact of the spaces being adjacent or not, if $\Omega_{pq} * \delta \leq d(p'', q'')$ then $\delta \leq d(p, q)$ and the friend in position q can be filtered out as not in proximity.

4.5 Experimental evaluation

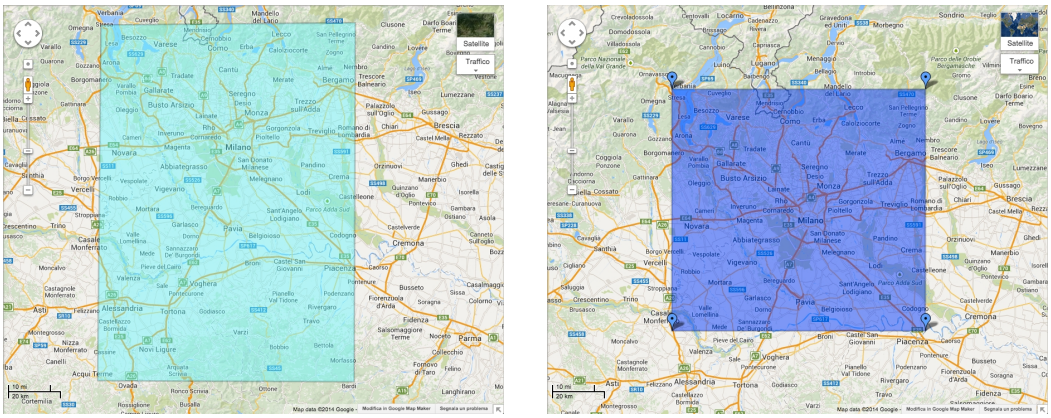
In this section we will discuss the experimental results of the three techniques (cartograms, recursive and greedy algorithms) capable to compute the uniformation of the users' distribution in the space. The aim is to evaluate the overall distortion that the three techniques produce from the non uniformed space S to the space S', in which users' distribution is uniform. Clearly, smaller is the distortion, higher is the filtering that can be performed and, consequently, higher are the performances of the overall technique described in Section 4.4.

4.5.1 Experimental settings

Figure 4.10: The GPWv3 italy data map [O5]

The data on the population distribution in the world is entirely taken from GPWv3 [O5], the same used in the location privacy attack described in Section 3.1. This dataset that provides density information by dividing the world into cells and providing the number of inhabitants for

each cell. Hence, we use the cells at the highest available resolution, i.e., 2.5 arc-minutes, that corresponds to a cell edge of about $5km$ at the equator. Since no public information is available at a higher resolution, in the current setup we assume that, within each cell, the population density is uniform.



(a) Milano-A (b) Milano-B

Figure 4.11: The Milano regions in which the experiments are conducted. Images retrieved on Sep, 2014 from http://maps.google.com

The data and the region we used in the experiments include two spaces S. The first is an area of the north-west region of Italy, including Milano, as shown in Figure 4.11(a): we refer to this region with the name "Milano-A"[1]. The area of the Milano-A region is approximately $13590km^2$. Another region we consider is a subregion of Milano-A that we call "Milano-B"[2]: the region is shown in Figure 4.11(b). Precisely, the region Milano-B is obtained by cutting the southern part of Milano-A so that the related inhabitants' distribution is less "heterogeneous": in this region we have the expectation that the uniformed regions, outputted by the algorithms, are less distorted. The overall area of the Milano-B region is $9050km^2$.

Subdivision	**Number of cells**	
	Milano-A	Milano-B
Coarse	900	600
Mid	3600	2400
Fine	14400	9600

Table 4.1: Subdivisions in S

In the experiments we variate the number of cells, according to Table 4.1. The fine subdivision counts 14400 and 9600 cells in the Milano-A and Milano-B regions, respectively, resulting in cells

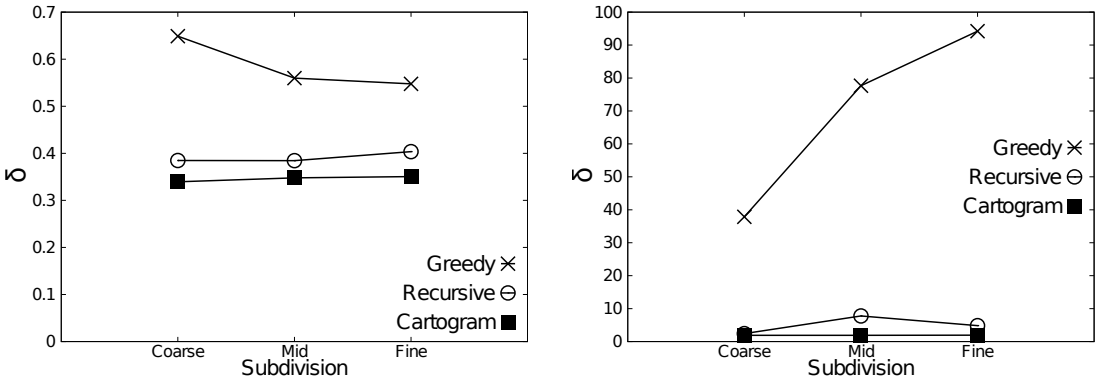
[1]Google Maps link: http://g.co/maps/utbr8
[2]Google Maps link: http://g.co/maps/8z6zq

with area nearly equal to $0.9km^2$. In the coarse subdivision there are 900 cells in the Milano-A area and 600 in the Milano-B, resulting in cells with area of about $15km^2$.

The Recursive and the Greedy algorithm have been implemented in Java, while for the cartograms we used the C library "Cart" by Mark Newman[3]. The experiments have been conducted on a $2.26GHz$ CPU with $4GB$ of main memory.

4.5.2 Results

The experiments that we will discuss in this section do not represent a full experimental evaluation. The aim was to identify the distortion that each algorithm causes to the space, so that we could have an idea of the data utility in the transformed space (as said before, smaller is the distortion, higher is the data utility in the transformed space). However, the preliminary results, as we will see, show that the distortion of the transformed space is too high to implement a good filtering. Furthermore, we will discuss only the results in the Milano-B region: the reason is that, as we will see, the results are not promising in this subarea, and, therefore, a discussion of the results we obtained in the Milano-A area, that are obviously worse than the ones we will present for the Milano-B region, is not significant.



(a) Avg distortion values

(b) Max distortion values

Figure 4.12: Results for Milano-B area, subdivisions varying

Given a distance d between two points in the original space and a distance d' of the corresponding points in the uniformed space, we define distortion as

$$\delta = \frac{(d' - d)}{d}$$

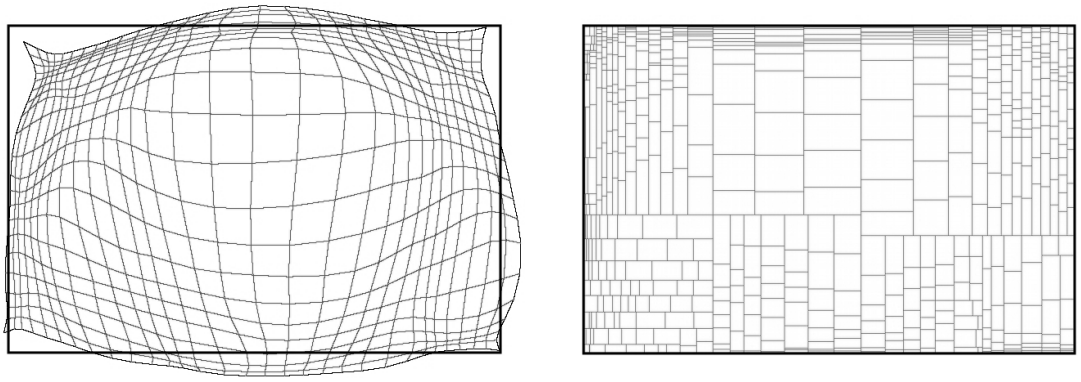
For each algorithm, we first compute the uniformed space S'. Then we randomly chose 100000 pair of points in S and we compute their distances in S' and tested the distortion δ. We performed the tests for both the regions Milano-A and Milano-B, but, as discussed before, we will present only the results in the Milano-B region.

In Figure 4.12(a) are shown the mean distortion for each subdivision in the Milano-B region. As we can see the Greedy algorithm is the one that shows the worst mean distortion. Indeed,

[3]http://www-personal.umich.edu/ mejn/cart/

for each subdivision considered, the mean distortion, i.e. the mean relative error, is over 0.5: this means that the percent error is over the 50%. Conversely, the Recursive[4] algorithm and the cartogram approach show similar results with a mean distortion under the value of 0.4.

Please note that the cartogram mean distortions are smaller than the ones of the Recursive algorithm for each subdivision considered. The reason underneath the better results of the cartograms approach is that with cartograms the space is uniformed in a continuous way, such that points that are "near" in the original space S, can not be too far in the uniformed space S'. In Figure 4.13 we have a graphic representation of how the cells are distorted by the cartograms approach and by the Recursive algorithm (the Greedy cells in S' are not shown since this technique does not alter the size of the cells): as we can see, the cartogram approach changes the sizes and the shape of the cells, while the Recursive algorithm changes the sizes of the cells, also shuffling them.



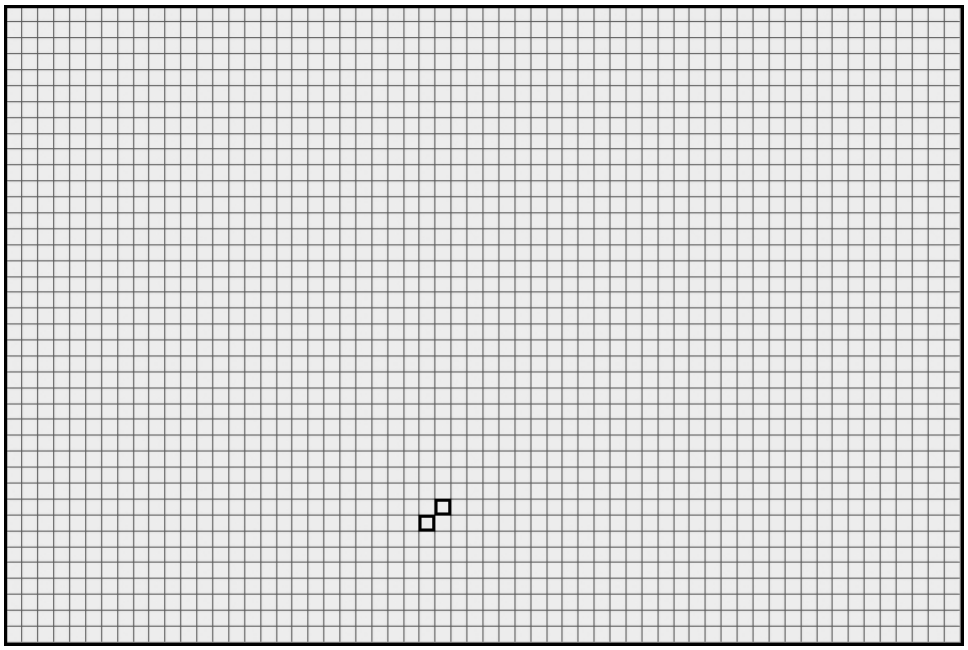
(a) Cartogram (b) Recursive algorithm

Figure 4.13: Cells' distortion in S' with the cartogram and Recursive algorithm

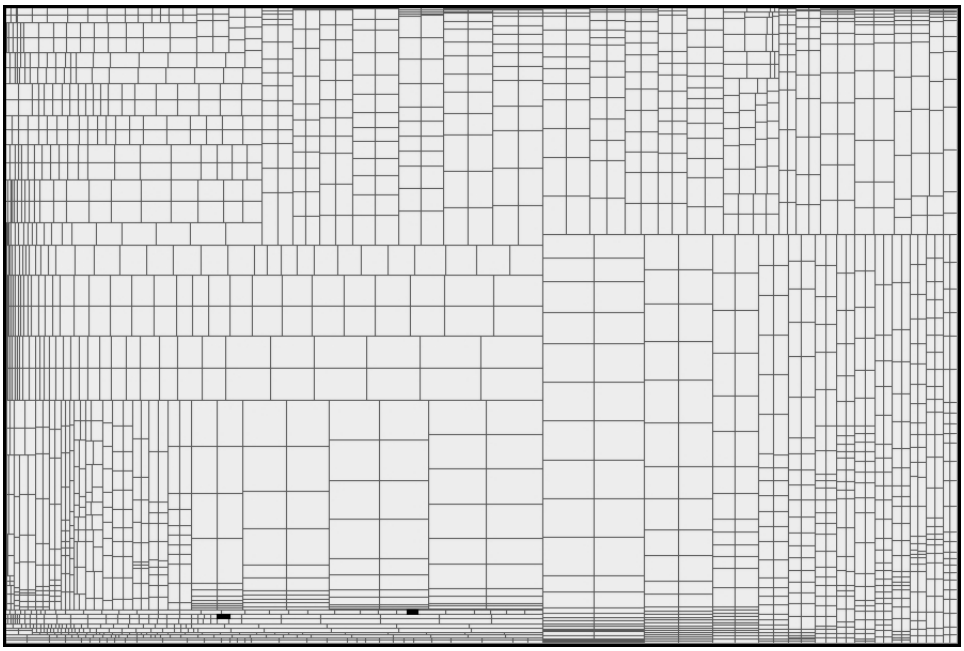
The fact that space is uniformed in a continuous way by the cartogram approach it is reflected also in the results shown in Figure 4.12(b), in which the maximum distortions values are represented. As we can see, compared with the other two, the maximum distortions are much higher for the Greedy algorithm whose values suggest that the algorithm applicability as a filtering technique is almost null. In general, also the maximum distortions for the Recursive algorithm can be high due to the fact that this algorithm can move far areas that are close in the original space. One example is the maximum distortion recorded for the Recursive algorithm for mid subdivision in the space Milano-B that is almost 8. In Figure 4.14 are highlighted the cells whose points showed such a large value of distortion. While the Recursive algorithm maximum distortions are much lower (the value is under 8 for all the subdivisions) compared to the Greedy ones, with the cartogram approach the distortion values are always under 2, as shown in Figure 4.12(b).

These results for the Milano-B area show that the overall distortion to obtain a uniformed space S' is remarkably high for the Greedy technique. The results for the Recursive algorithm and, in particular, for the cartograms approach are more encouraging.

[4]The results of this section for the Recursive algorithm make use of the *findCut* algorithm (see Algorithm 5). Since the results obtained with the $findCut - S$ (see Algorithm 6) are very similar, we omit their discussion here.



(a) Space S



(b) Space S'

Figure 4.14: Two adjacent cells in S are far in the space S' computed by the Recursive algorithm in Milano-B region, mid subdivision

4.6 Summary

In this chapter we have proposed and analysed a new technique for distance-based services, while protecting users' location privacy when the adversary's knowledge includes both the observation of the distances between some pairs of users and the user's distribution in the considered space. This technique is composed of two phases: in the first phase a filtering technique is executed, such that some of the users can be discarded, since they are with certainty not in proximity with u. For the remaining users, a cryptographic technique as [44], is executed to infer which users are really in proximity with the user u.

In this chapter we focus mainly on the first step, in which we used a distance preserving transformation, whose safety is forced through a uniformation of the users' distribution in the space. We propose and present two algorithms capable of uniforming the users' distribution and we experimentally evaluate them, together with a well known technique: the cartograms. The results show that for both the algorithms and for the cartograms approach the distortion needed to uniform the users' distribution in the space is high, hence granting only a limited filtering potential.

5. Privacy protection by generalisation

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. As discussed in Chapter 2, an extensive literature exists about location privacy and protection techniques [6, 58]. Spatial and temporal generalisation is a common approach that has also been implemented in prototypes and applications.

The term "generalise" is quite generic: in the Oxford English Dictionary[1] this term has 8 different meanings. The one we are referring to in this chapter is "To make (a thing) more general or indefinite; to remove or diminish the distinctive features or details": the aim of the generalisation techniques is to enlarge the user's position such that its "distinctive features or details" are less sensitive. Indeed, spatial generalisation decreases the precision of location information, hence introducing uncertainty about the actual position of the considered moving object within the reported geographical area. Temporal generalisation introduces uncertainty about the precise time of presence of the moving object 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 chapter we focus on the generalisation of a source point representing the user's exact location and time of presence in a spatial or spatio-temporal region. A crucial issue for this approach is deciding how large the generalised region should be (in both space and time) 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 generalised region is actually dependent on the context, including the individual's privacy

[1] http://www.oed.com/

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., [18, 29, 41]), have the goal to identify *the smallest* region 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 generalised region 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 generalisation solutions, not focused on identity privacy but more on hiding the presence of an individual in a potentially sensitive place (e.g., [10]) have a different optimisation criteria for the dimension of their generalised regions. 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.

Many privacy preserving techniques make use of spatial/spatio-temporal generalisation, but the generalisation it is often sketched out, not described comprehensively (as we'll discuss in Section 5.1 in more details). In this chapter, we formalise two spatial granularities and one spatio-temporal generalisation technique that can be used by any privacy preserving techniques that needs to perform a location's generalisation. Please note that our proposed generalisation methods are specifically designed for location privacy techniques but are flexible enough to be exploited for any general purpose technique in which a position needs to be generalised. Since we are proposing generalisation tools that are designed to be implemented in location privacy techniques, but cannot be considered comprehensive privacy preserving techniques, an evaluation of data utility of these generalisations it is not suitable. Instead, we study the costs of their computation, both formally and experimentally.

In Section 5.1 we will discuss other spatial and spatio-temporal generalisations proposed in the literature. In Section 5.2 we will present two spatial generalisation techniques, $\mathscr{G}$onio and $\mathscr{A}$equus, whose results have been published in [5]. Finally, the Safebox spatio-temporal generalisation is formalised in Section 5.3, and it is based on the following publication: [39].

5.1 Related generalisation techniques

Some contributions in the literature adopted spatial and spatio-temporal generalisation 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 [25]. 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 generalisation function is known to the adversary. This problem, called "inversion" or "reciprocity", has been addressed in [29, 41]. These two papers propose two analogous formal properties of the generalization function. Intuitively, a generalisation function $\mathscr{G}$ meets these properties if each point contained in any generalised region r computed by $\mathscr{G}$ is generalised to r

itself. The two papers prove that, if a generalisation algorithm meets this property, then it is safe with respect to the inversion problem. In the Safebox proposal we propose a different property (see Section 5.3.2) that intuitively states that a generalised region r computed by a generalisation function $\mathscr{G}$ should contain at least k objects that, if used as source points, are generalised 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) generalise 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 generalisation algorithms with respect to the inversion problem. This new definition is required by our *BottomUp* algorithm that does not meet the properties defined in [29, 41] but instead meets the new property defined in this chapter and hence it is safe with respect to the inversion problem. The main problem with the generalisation algorithms proposed in [29, 41] 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 personalised value of k [18], the solution by Abul et al. that makes it possible to anonymise 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 [43].

In the contribution by Ghinita et al. the objective of the spatial generalisation is not to provide anonymity rather to decrease the sensitivity of the location information [20]. The technical problem is that, given two generalised 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 generalised spatio-temporal regions should be created, which instead is the focus of the SafeBox contribution. We believe that extending the Safebox solution with a technique like the one presented in [20] could be an interesting future work.

Other contributions share the same objective of decreasing the sensitivity of the location information [1, 20, 42]. Some of them are specifically designed for the so-called friend-finder services [44, 57, 62] 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 generalised 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 [3, 5, 46]. 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 generalised area [3]. Unfortunately both techniques suffer from the inversion problem and hence are safe only if the generalisation technique is not known to the adversary. Our previous work has the same problem [5]. Indeed in [5] we define three spatial granularities to support privacy preservation. In particular the $\mathscr{I}ncognitus$ 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 5.3.3, 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 [10]. This is an innovative approach and it has two main differences with respect to ours. First, the generalisation applies to the entire map and is computed off-line. Vice versa, in the SafeBox solution we generalise the location of the user on-the-fly, so the generalisation function can use dynamic information. The second difference is that in the solution by Damiani et al. a user's location is generalised 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 solutions we propose generalisation functions that can be used to generalise requests from every location.

5.2 Gonio and Aequus granularity families

Spatial granularity has often been used in the literature as formal tool to partition a spatial domain into a set of sub-regions called "granules" [7, 11, 49]. In recent years spatial granularities have been extensively used, implicitly or explicitly, in several papers in the field of privacy preservation in presence of location information. A comprehensive survey on privacy preserving techniques for private queries is presented in [19].

A common problem in the contributions mentioned above is that spatial granularities are used, often implicitly, as a tool, without extensively describing their properties and basic operations.

In this section we provide a general definition of spatial granularities and we identify two basic operations and two properties, "non-overlapping" and "space-covering", that are particular relevant when using spatial granularities as a tool to protect privacy.

We also introduce the concept of "family of granularities", representing set of granularities with similar characteristics but different size of the granules. The main contribution of this section consists in the definition of two family of granularities, specifically designed to be used in privacy preserving applications. The two families "Gonio" (labeled $\mathscr{G}$) and "Aequus" (labeled $\mathscr{A}$) are designed for the "MUR approach". For each family of granularities we show how to efficiently compute the two basic operations and we prove that they respect the "non-overlapping" and "space-covering" properties.

5.2.1 Problem Formalization

In this section we consider the entire world as the reference spatial domain S. We use standard latitude-longitude coordinates, hence: $S = (-90, 90) \times [-180, 180)$. We define a *spatial granularity* (or simply "granularity") G as a mapping from a subset of $\mathbb{N}$ to S. For each $G(i)$ we call i an *index* and, if $G(i)$ is defined, we call $G(i)$ a *granule*, denoted as g. If g is a granule of a granularity G, we denote it with $g \in G$.

We design granularities for privacy-aware applications and we identify two relevant properties for this type of applications. First, privacy protection should be provided everywhere, hence we need granularities that are "space-covering" meaning that any point of the spatial domain should be covered by the granules, as formalised in Definition 5.2.1.

Definition 5.2.1 A granularity G is *space-covering* if, for every point $p \in S$, there exists one granule $g \in G$ such that $p \in g$.

The "non-overlapping" property, formalised in Definition 5.2.2, is related to the fact that it should be possible to uniquely identify a granule covering a point in the space. In other words, there should be no intersection between the granules of a given granularity. The lack of the "non-overlapping" property has been noted to negatively impact the privacy guarantees of the "MUR approach" [10].

Definition 5.2.2 A granularity G is *non-overlapping* if, for every pair of $g_1, g_2 \in G$, with $g_1 \neq g_2$, $g_1 \cap g_2 = \emptyset$

In this section we are interested in defining families of granularities. The intuition is that all the granularities belonging to a family share the same characteristics, but have different "size" of the granules. Formally, a *family of granularities* is a mapping associating a granularity to an element of $\mathbb{N}$ that we call "level". Given a family of granularities $\mathscr{F}$, we denote with $\mathscr{F}^l$ the granularity G having level l in the family $\mathscr{F}$.

In the following we characterise each granularity G by defining $G(i)$ for each index $i \in \mathbb{N}$ and we show how to efficiently compute this function. The "inverse" function $G[p]$ returns, for each $p \in S$, the index i such that $p \in G(i)$. Note that, since we consider "non-overlapping" and "space-covering" granularities, $G[p]$ is always defined, in the sense that for each point $p \in S$ there exists exactly one granule g of G such that $p \in g$. Once it is known how to compute $G(i)$ for every index i, it is possible, in theory, to compute $G[p]$ by computing $G(i)$ for each index i and choosing the one such that $p \in G(i)$. However, this is impractical and in the following of this section we describe how to efficiently compute $G[p]$.

5.2.2 The Gonio granularities

The $\mathscr{G}$ granularity family partitions the spatial domain S into rectangular granules all sharing the same angular dimension. The result on a map of such a partition is the same as the one performed by the well-known Mercator projection, that is the standard projection for nautical purposes. The Mercator projection distorts the size and shape of large objects, as the scale increases from the Equator to the poles, where it becomes infinite, as shown in Figure 5.1.

The level of the granularity represent the number of granules, where $\mathscr{G}^0$ has only one granule which comprises the entire spatial domain while a granularity of level l has 4^l granules, organized in a grid of 2^l granules along each dimension. This is intuitively analogous to a balanced quadtree as shown in Figure 5.2. In the following we will refer to this hierarchical structure as the "granularity tree".

As shown in Figure 5.3, the granule with index zero has the bottom-left corner in $90°S$, $180°W$, while the others granules are enumerated from left to right and from bottom to top. Since a granularity $\mathscr{G}^l$ has 2^l granules along each dimension, the angular size of each granule is $180°/2^l$ on the vertical dimension and $360°/2^l$ on the horizontal dimension.

The regular structure of $\mathscr{G}$ granularities allows to define the position of each granule in terms of its column X_i and row Y_i. This is an alternative format to index granules that will be useful in the following of this presentation. Clearly, it is possible to convert from the "single-index" format

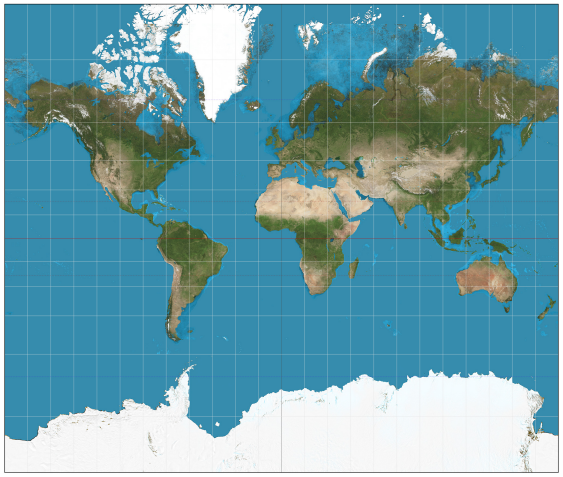
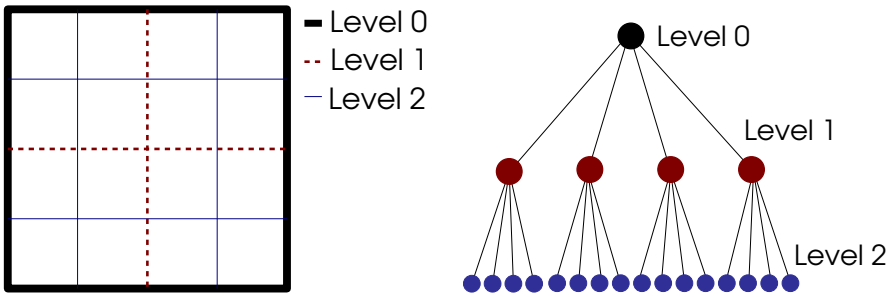


Figure 5.1: Mercator projection map. Image by Strebe - Own work. Licensed under CC BY-SA 3.0



(a) Projection plane (b) granularity tree

Figure 5.2: Different granularity levels with $\mathscr{G}$ granularity family on the plane and their representation in terms of granularity tree

to the "double-index" format in constant time using these formulas:

$$X_i = i \mod 2^l \qquad Y_i = \lfloor \frac{i}{2^l} \rfloor$$

$$i = X_i + 2^l \cdot Y_i \tag{5.1}$$

We now formally characterise a granularity $\mathscr{G}^l$ through the specification of $\mathscr{G}^l(i)$ for a generic index i.

Definition 5.2.3 Let l and i be two non-negative integers. $\mathscr{G}^l(i)$ is undefined for $i \geq 4^l$. Otherwise

$$\mathscr{G}^l(i) = [minLon, maxLon) \times [minLat, maxLat)$$

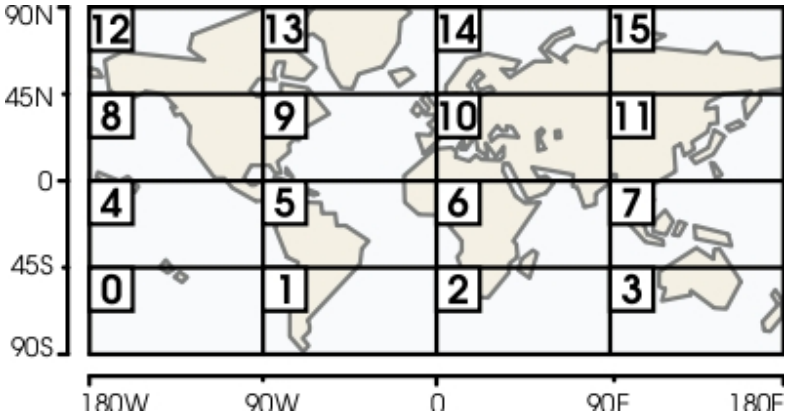


Figure 5.3: $\mathscr{G}^2$ shown on the world

where:

$$minLon = X_i \cdot \left(\frac{360}{2^l}\right) - 180 \tag{5.2}$$

$$maxLon = (X_i + 1) \cdot \left(\frac{360}{2^l}\right) - 180 \tag{5.3}$$

$$minLat = Y_i \cdot \left(\frac{180}{2^l}\right) - 90 \tag{5.4}$$

$$maxLat = (Y_i + 1) \cdot \left(\frac{180}{2^l}\right) - 90 \tag{5.5}$$

Note that, in order to guarantee the "space-covering" and "non-overlapping' properties, Definition 5.2.3 specifies that any point on the minimum coordinates (left-bottom of the granule) is part of the granule, while any point on the maximum coordinates (right-top of the granule) is not part of the granule.

We now show how to compute $\mathscr{G}^l[p]$ in constant time. That is, how to find the index i of the granule containing the point p (expressed in the $\langle lat, lon \rangle$ format). The column and row positions are computed as the proportion of longitude and latitude (respectively) with respect to the size of the horizontal and vertical domain (respectively). Formally:

$$X_i = \left\lfloor \frac{2^l \cdot (lon + 180)}{360} \right\rfloor \tag{5.6}$$

$$Y_i = \left\lfloor \frac{2^l \cdot (lat + 90)}{180} \right\rfloor \tag{5.7}$$

Consequently the $\mathscr{G}^l[p]$ can be defined as follows:

$$\mathscr{G}^l[p] = \left\lfloor \frac{2^l \cdot (lon + 180)}{360} \right\rfloor + 2^l \cdot \left\lfloor \frac{2^l \cdot (lat + 90)}{180} \right\rfloor \tag{5.8}$$

Given the above definitions, the functions $\mathscr{G}^l(i)$ and $\mathscr{G}^l[p]$ can be clearly computed in constant time.

It is also easily seen that $\mathscr{G}$ granularities are "space-filling" and "non-overlapping", as shown in Properties 5.2.1 and 5.2.2.

Property 5.2.1 For every level l, $\mathscr{G}^l$ is space-covering.

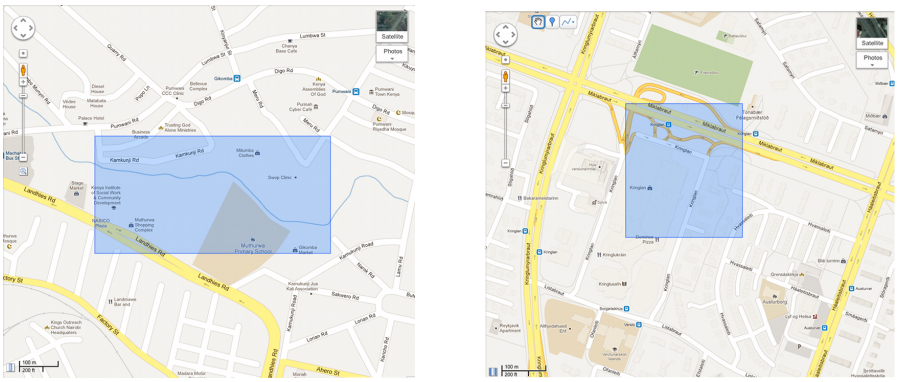
Property 5.2.2 For any level l, $\mathscr{G}^l$ is non-overlapping.

Proofs of these two properties are reported in Appendix A.

5.2.3 The Aequus granularities

As specified in Section 5.2.2, the granules of a $\mathscr{G}$ granularity have the same angular dimension. Thus, granules at different latitudes covers areas of different size. Consider Example 5.1.

■ **Example 5.1** We consider two granules of $\mathscr{G}^{16}$, one located in Nairobi, Kenya (-1.2877, 36.8372) and the other in Reykjavik, Iceland (64.1324,-21.8934). The granule positioned in Nairobi has an area more than two times the area of the granule calculated in Reykjavik, as shown in Figure 5.4.



(a) Nairobi-186314.2m^2 (b) Reykjavik-81281.7m^2

Figure 5.4: Example of granules of $\mathscr{G}^{16}$ with different areas. Images retrieved on Feb 15, 2013 from http://maps.google.com

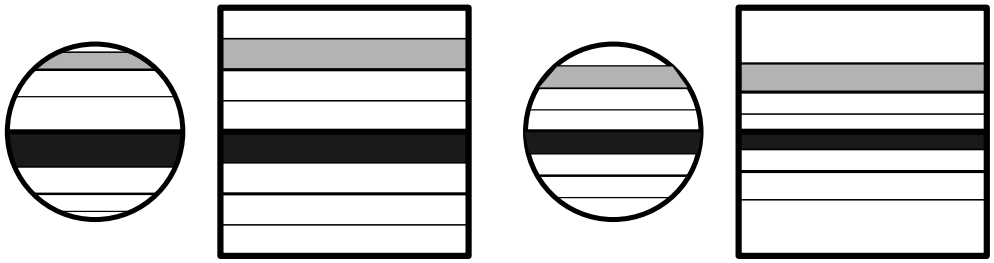
■

Different dimensions of the granules are undesirable in some applications. For example, in the "MUR approach" to privacy preservation, a $\mathscr{G}$ granularity yields different level of privacy in different regions of the world. In this section we present the $\mathscr{A}$ granularities, an extension of $\mathscr{G}$ granularities, with the following property: all granules of $\mathscr{A}^l$ have the same area, independently from their latitude.

Technically, $\mathscr{A}$ granularities are specified like $\mathscr{G}$ granularities; the only difference is in the computation of the horizontal boundaries of the granules. To understand the intuition behind our solution, we suggest to think about a $\mathscr{G}$ granularity as a set of horizontal stripes that are vertically "cut" to create the granules. In this view, each stripe is divided into the same number of granules

all having the same width. Since we are considering Earth as our reference space, these "stripes" are actually spherical segments[2].

In a $\mathscr{G}$ granularity the spherical segments have the same angular height, so they appear as having the same area in the Equirectangular projection, but they actually have different areas (see Figure 5.5(a)). This clearly leads the granules to have different areas. Vice versa, we define $\mathscr{A}$ granularities with spherical segments having different angular height but with the same area (see Figure 5.5(b)). Consequently, the granules result to have the same size, as we will prove.



(a) Spherical segments in $\mathscr{G}^3$ (b) Spherical segments in $\mathscr{A}^3$

Figure 5.5: Intuitive view of $\mathscr{G}$ e $\mathscr{A}$ granularities

We now define the family of $\mathscr{A}$ granularities by specifying $\mathscr{A}^l(i)$ for a generic index i.

Definition 5.2.4 Let l and i be two non-negative integers. $\mathscr{A}^(\mathrm{i})$ is undefined for $i \geq 4^l$. Otherwise

$$\mathscr{A}^l(i) = [minLon, maxLon) \times [minLat, maxLat)$$

where *minLon* and *maxLon* are defined as in Equations 5.2 and 5.3 and:

$$minLat = \arcsin\left(1 - \frac{2Y_i - 2}{2^l}\right) \tag{5.9}$$

$$maxLat = \arcsin\left(1 - 2\,\frac{Y_i}{2^l}\right) \tag{5.10}$$

To understand the idea behind the above definition, we first recall that the area of a spherical segment with height h is equal to $2\pi\mathscr{R}h$ where $\mathscr{R}$ is the radius of the sphere. Using the above definitions of *minLat* and *maxLat*, the height of a spherical segment containing a generic granule $\mathscr{A}^l(i)$ is equal to

$$h = \mathscr{R} \cdot [sin(maxLat) - sin(minLat)] \tag{5.11}$$

$$= \mathscr{R} \cdot \left[1 - 2 \cdot \frac{Y_i}{2^l} - \left(1 - 2 \cdot \frac{Y_i}{2^l} - \frac{2}{2^l}\right)\right] = \frac{2\mathscr{R}}{2^l} \tag{5.12}$$

[2]Here and in the following we approximate Earth with a sphere.

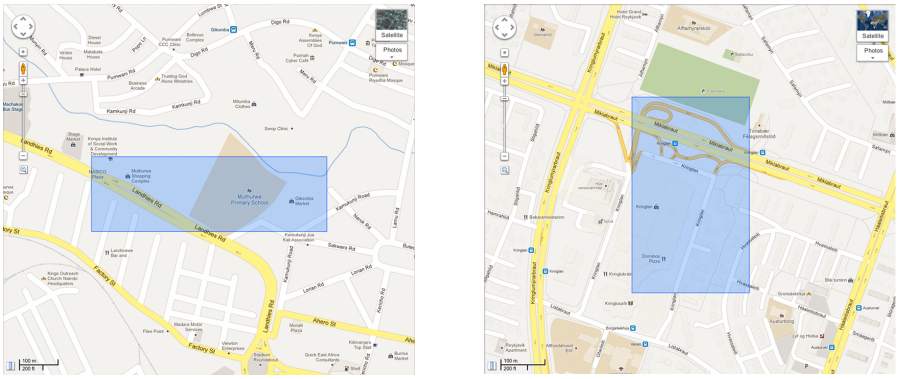
This spherical segment has consequently an area A_{seg} that is independent from i and that is actually equal to $A_{seg} = \frac{4\pi\mathscr{R}^2}{2^l}$.

Property 5.2.3 follows from the above results and it shows that all granules in a given granularity $\mathscr{A}^l$ have the same area.

Property 5.2.3 Let G be the granularity $\mathscr{A}^l$. Then each granule g of G has an area equal to:

$$\frac{A_{seg}}{2^l} = \frac{4\pi\mathscr{R}^2}{4^l}$$

■ **Example 5.2** Any granule of $\mathscr{A}^{12}$ has an area of about 30km^2, while granules of $\mathscr{A}^{16}$ have an area of about 0.12km^2. Indeed, compare Figure 5.4 with Figure 5.6 that shows two granules of $\mathscr{A}^{16}$ in Nairobi, Kenya and Reykjavik, Iceland, respectively. Differently from what observed in Example 5.1, in this case the two granules have the same area.



(a) Nairobi, Kenya (b) Reykjavik, Iceland

Figure 5.6: Example of granules of $\mathscr{A}^{16}$ with same areas. Images retrieved on Feb 15, 2013 from http://maps.google.com

The $\mathscr{A}^l[p]$ operation (for a given point $p = \langle lat, lon \rangle$) is analogous to the same operation for $\mathscr{G}$ granularities, with the only difference in the computation of Y_i that is defined as:

$$Y_i = \left\lfloor \frac{2^l \cdot (1 - \sin(lat))}{2} \right\rfloor$$

Consequently:

$$\mathscr{A}^l[p] = \left\lfloor \frac{2^l \cdot (lon + 180)}{360} \right\rfloor + 2^l \cdot \left\lfloor \frac{2^l \cdot (1 - \sin(lat))}{2} \right\rfloor$$

Similarly to $\mathscr{G}$ granularities, also $\mathscr{A}$ granularities are "space-filling" and "non-overlapping". Proofs are analogous to the case of $\mathscr{G}$ and are omitted.

■

5.3 SafeBoxes

Spatial and temporal generalisation emerged in the literature as a common approach to preserve location privacy. However, existing solutions have two main shortcomings. First, spatio-temporal generalisation 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 generalisation 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., moving positions or changing properties of the objects) and needs to be acquired from external services (e.g., Google Maps), that typically allow a limited number of localised queries.

In this section 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 generalisation is compatible with the use of online services, it supports generalisations 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 generalised regions.

5.3.1 Problem and contribution

As discussed in Chapter 5.1, spatial and temporal generalisation is a common approach that has also been implemented in prototypes and applications. In this chapter we call *SafeBox* the spatio-temporal region used to generalise a source point representing the user's exact location and time of presence.

The Problem

Each of the proposed techniques discussed before is somehow specialised to optimise the generalisation with respect to 1) a precise use case and to 2) the specific semantics of privacy preservation (counting users, venues, categories of venues, ...). Hence, one problem we would like to address in this chapter is to have a generalisation 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 [29, 41], most of the proposed techniques for spatial generalisation, 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 generalisation 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 generalisation 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 chapter 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 generalisation 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 section are the following:

- By providing a general notion of object counting function supporting different semantics we capture in a single problem formalisation several location privacy problems previously considered in the literature and enable capturing privacy preferences not considered in previous generalisation approaches.
- We design a generalisation 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 generalisation 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 section is structured as follows. In Section 5.3.2 we formalise the privacy problem and we define the two semantics of the counting function that we use in our experimental evaluation. Section 5.3.3 presents the new *BottomUp* algorithm and compares it with one following the more traditional top-down approach. The experimental evaluation is reported in Section 5.3.4, and Section 5.4 concludes the chapter.

5.3.2 Problem Formalization

We address the problem of *generalising* 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 generalisation 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.

Adversary model

The generalisation techniques proposed in this contribution can be used to protect users' privacy in different system architectures. Indeed, the generalisation function can be computed either by the user's mobile client or by a trusted generalisation 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 generalised location is disclosed.

The "adversary" is any entity that can potentially have access to the generalised spatio-temporal location. 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 chapter we assume that the adversary has the knowledge to do the inversion attack that, as we formalise in the following of this section, requires the knowledge of the generalisation algorithm, its input parameters (with the exception of the source point) and the counting function. In a real-world generalisation service the generalisation 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 5.3). Finally, for what concerns the knowledge of the counting function, in some case this can be public information (as in Example 5.3) and, if not, it can be inferred, possibly introducing some approximation in the computation, like in Example 5.3.

■ **Example 5.3** Alice uses a privacy-aware LBS. Before issuing any service request, the client generalises 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 generalised 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 generalised to region smaller than A. Hence the value of k is either 5 or 6.

Now, suppose that the generalisation 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 chapter we consider the spatio-temporal generalisation 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 generalised spatio-temporal region [20]. Since at the moment our generalisation 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 [43]. In this case, if it is possible to "link" different generalised 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 generalisation 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 generalised regions can be associated to the same user for example by artificially changing the IP address used in the corresponding service requests.

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 *generalising* any point p (called "source point") into a three-dimensional area with certain properties. Formally, a *generalisation function* $\mathscr{G}$ 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 generalisation 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 5.3.2 we report two examples of its semantics. Note that the source point being generalised 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 chapter assume only that the counting function is monotonic with respect to the areas.

Definition 5.3.1 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 5.3.2 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 [29, 41], even if a generalisation function returns a SafeBox according to Definition 5.3.2, an adversary may still be able to rule out some of the objects considered in the counting if he knows the generalisation function itself, since he may re-apply the generalisation function to each candidate source point and compare the result with the area that has been released. In principle, by knowing the generalisation function the adversary may also be able to identify the source point p though the so called *inversion attack* [41]. Consider Example 5.4.

■ **Example 5.4** Let's consider Figure 5.7: 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 generalisation is to have a SafeBox with at least 8 objects. Let's consider a naive generalisation 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 generalised to $A_3 \cup A_4$.

Now, consider an adversary that knows the generalisation 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 generalisation would be A_2 itself. Consequently the adversary infers that the source point is in A_1 that, however, is not a SafeBox.

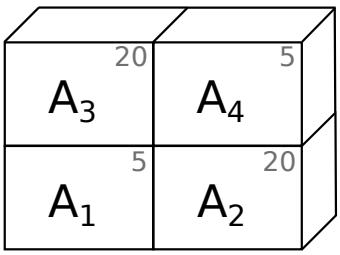


Figure 5.7: Spatio temporal domain partitioned into four areas.

■

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

Definition 5.3.3 Given an area A and a generalisation function $\mathscr{G}$, the *inversion* function I is

defined as:

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

When no confusion arises, we simply denote $I(A)$ omitting the generalisation function.

We can now define the notion of safety for generalisation functions.

Definition 5.3.4 Given a non-negative integer k and a counting function Ω, a generalisation function $\mathscr{G}$ is *safe* if, for each spatio-temporal point p such that $\mathscr{G}(p)$ is defined, it holds that:

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

■ **Example 5.5** Let's continue with Example 5.4. We show that the generalisation function is not safe according to our model. Indeed, for any point p in A_1, the generalisation of that point is $A_1 \cup A_2$. The inversion of $A_1 \cup A_2$ is A_1 (indeed the generalisation of any point in A_2 is A_2 itself). Consequently, $\Omega(I(\mathscr{G}(p))) = \Omega(A_1) = 5 < 8$. Consequently, by applying Definition 5.3.4, the generalisation function is not safe, in accordance with the intuition presented in Example 5.4.

Let's now consider a different generalisation function that generalizes any source point of A_2 in A_2 and similar for A_3. Also, the generalisation of any source point in A_1 or A_4 is the entire spatio-temporal domain (i.e., A). This is actually a safe generalisation function. Indeed, for any source point p in A_2 it holds that its generalisation 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 generalisation yields A. In this case $I(A) = A_1 \cup A_4$. Hence we have that $\Omega(I(\mathscr{G}(p))) = \Omega(A_1 \cup A_4) = 10 \geq 8$. 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 5.3.1 Any safe generalisation function $\mathscr{G}$ returns a SafeBox for any source point p such that $\mathscr{G}(p)$ is defined.

Proofs of formal results are reported in Appendix A.

Supporting different semantics for the counting function Ω

As specified in Section 5.3.2, 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.

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 5.3.5 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 5.3.6 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 5.6** 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 5.3.2 The counting function $\Omega()$ with appearance semantics is monotonic.

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 formalise 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 5.3.7 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 5.7** Suppose we are computing $\Omega()$ for the centre 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 5.3.7 by the predicate $loc(o,t) \in A_S$. The number of persistence intervals intuitively gives a value for a "temporal obfuscation" metrics. ■

Property 5.3.3 The counting function $\Omega()$ with persistence semantic is monotonic.

5.3.3 SafeBox computation

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

The *TopDown* algorithm starts by considering the entire space and time (the "top") and then "moves down" from the root of the generalisation 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 generalisation 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 generalisation 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.

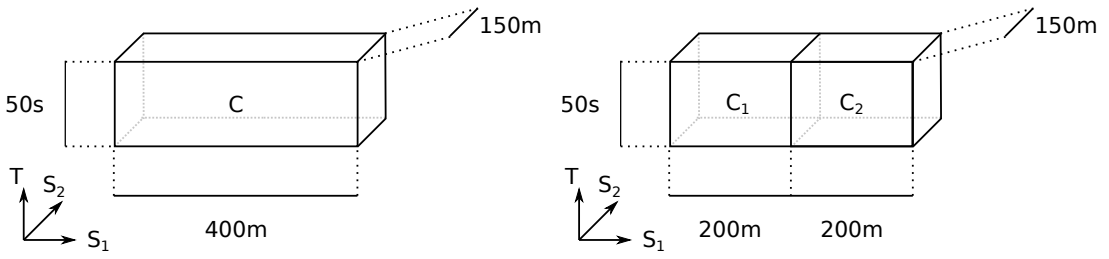
In the following of this section we first formalise the generalisation tree in Section 5.3.3, and then we present the *TopDown* and *BottomUp* algorithms in Sections 5.3.3 and 5.3.3, respectively.

Data structure: the generalisation tree

A *generalisation 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 generalisation 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 5.8.

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 5.8.



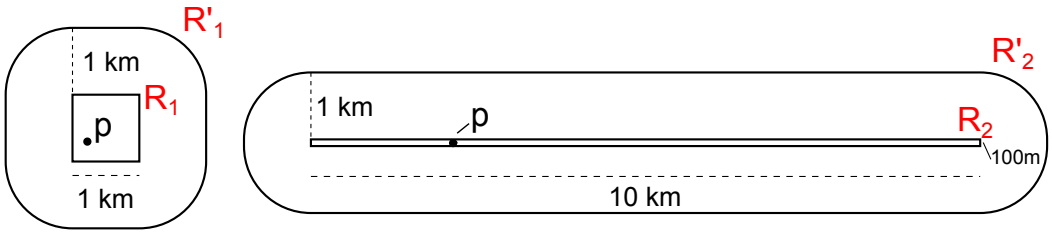
(a) The ST-cell c (b) The two children of c

Figure 5.8: A ST-cell c and its two children c_1 and c_2

■ **Example 5.8** 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 generalised locations from the users.

To protect Alice's privacy, her client always generalises 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 generalisation algorithms: one returns a squared region R_1, the other the stretched region R_2 (see Figures 5.9(a) and 5.9(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 generalisation to R_2 incurs into larger communication costs and larger computational costs both on the client and on the server. ■



(a) Squared. (b) Stretched.

Figure 5.9: Comparison between a squared generalised region and a stretched one.

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 5.8. More application-oriented criteria are needed to decide when to generalise 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.

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

Definition 5.3.8 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 formalise how children are constructed.

Definition 5.3.9 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.

The *TopDown* algorithm

The intuitive idea behind the *TopDown* algorithm is to traverse the generalisation 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 5.3.4, this termination condition makes this generalisation function safe. The basic version of the algorithm is shown in Algorithm 9.

■ **Example 5.9** Consider a generalisation tree like the one in Figure 5.10. 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 $k = 5$.

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.

■

Algorithm 9 *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

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 generalisation function. Vice versa, in all other cases (i.e., when $\Omega(S \times T) \geq k$), *TopDownBasic* can always find a SafeBox.

Algorithm 9 has a computational issue: each time the algorithm moves one level down in the generalisation 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 optimised version *TopDown* that returns the same result of *TopDownBasic* (see Theorem 5.3.5) but that avoids computing Ω for two overlapping regions.

Algorithm 10 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 generalisation 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 5.3.1) $\Omega(c_1) \geq k$. 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 generalisation 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

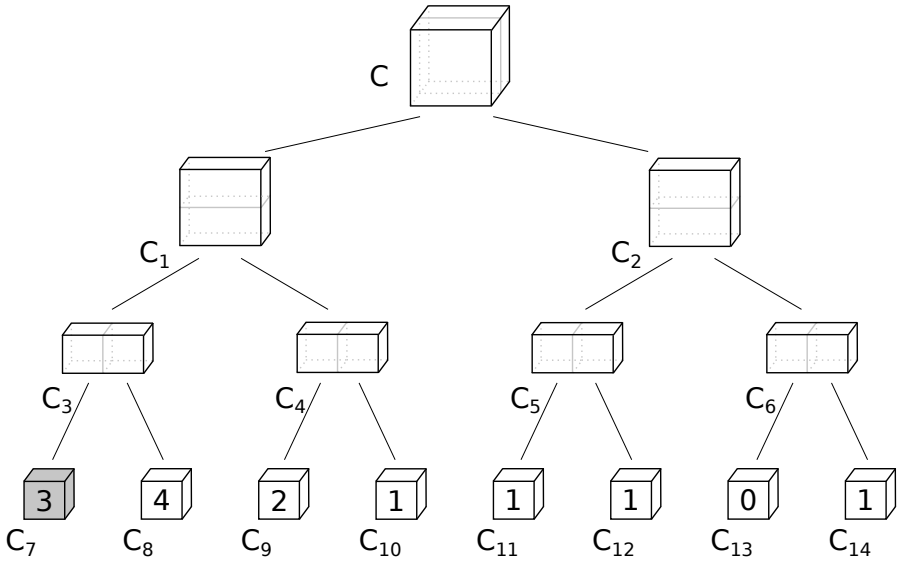


Figure 5.10: Example of generalisation tree

algorithm explicitly compute this condition and it returns c if the condition is met, the parent of c otherwise.

The *BottomUp* algorithm

The *BottomUp* algorithm processes the generalisation 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 *Incognitus*, in a preliminary investigation on this problem [5]. Unfortunately, the result obtained with this approach is indeed a *SafeBox*, but the generalisation function is not safe according to our definition, as shown in Example 5.10.

■ **Example 5.10** Consider the generalisation tree reported in Figure 5.11, a value of k equal to 4 and a bottom up generalisation algorithm that, starting from the leaf ST-cell containing the source point, continues to generalise if the value of Ω for the current ST-cell is less than k.

The generalisation 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 5.4. Intuitively, by observing a generalisation to C, an adversary may exclude as a candidate source point any point in C_{21} (since it would be generalised 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.

■

Algorithm 10 *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**

The *BottomUp* algorithm we propose in this chapter 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 11 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 generalisation 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 5.3.4).

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 generalisation 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).

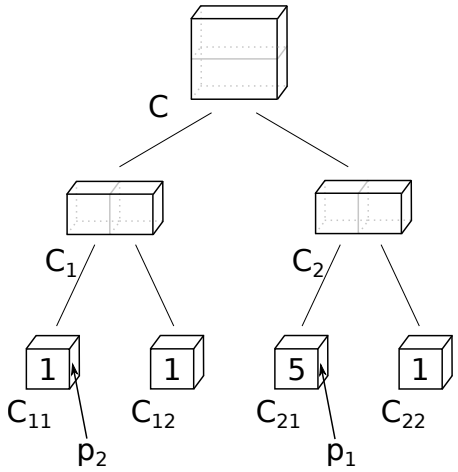


Figure 5.11: Unsafety of an intuitive bottom-up strategy

Algorithm 11 *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 5.11** Let's consider again the generalisation tree in Figure 5.10. As in Example 5.9 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 5.10). 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 5.10. 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 generalisation 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 5.11 we just gave an intuitive motivation for the result of the inversion computation. Two problems arise when trying to directly apply Definition 5.3.3 to compute $I(c)$ for a ST-cell c when the generalisation 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 5.3.3, 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 generalised 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 12. The idea is that, instead of directly computing Definition 5.3.3, which applies to any generalisation function, we adopt Procedure 12 that is specific for *BottomUp* and that, in the computation, does not require to recursively call *BottomUp* itself.

Procedure 12 *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: $res = Residuals(c, \alpha, \Omega, k)$;
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: $A = Residuals(c', \alpha, \Omega, k)$
5: **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 12) that uses *Residuals*, a recursive procedure that computes the set of all points whose generalisation 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*) generalise to c. Otherwise, (i.e.,

$\Omega(res) < k$), the counting condition in *BottomUp* for the points in *res* is not satisfied and hence the generalisation 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 generalisation 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 $Residuals(c', \alpha, \Omega, k)$ is less than k. If this is the case, then every point in the result of $Residuals(c', \alpha, \Omega, k)$ is added to the result that is being computed for c (Line 5). Otherwise, (i.e., $\Omega(Residuals(c', \alpha, \Omega, k)) \geq k$) every point in c' generalises to a ST-cell smaller than c, hence no point of c' contributes to the result.

■ **Example 5.12** Let's consider Example 5.11 that illustrates the application of the *BottomUp* algorithm to the generalisation tree in Figure 5.10, 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 $BottomUpInversion(C_3)$ [3]. 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 5.11.

■

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.

Safety

In order to show the correctness of the algorithms we have to prove that both *TopDown* and *BottomUp* compute a safe generalisation function. For *TopDown*, we first show that *TopDownBasic* is a safe generalisation function and then we show that *TopDown* computes the same result as *TopDownBasic* (hence *TopDown*is a safe generalisation function). This is formally stated in Theorems 5.3.4, 5.3.5.

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

Theorem 5.3.4 The *TopDownBasic* algorithm computes a safe generalisation function.

Theorem 5.3.5 For any source point p, any time influence parameter α, any $\Omega()$ function, and any the integer value k it holds that $TopDownBasic(p, \alpha, \Omega, k) = TopDown(p, \alpha, \Omega, k)$.

Before presenting the formal result for *BottomUp* in Theorem 5.3.7, we first present Property 5.3.6 that formally states that *BottomUpInversion* actually computes the inversion for *BottomUp*.

Property 5.3.6 Let $\mathscr{G}$ be the generalisation 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 $I_{\mathscr{G}}(c)$.

Theorem 5.3.7 The *BottomUp* algorithm computes a safe generalisation 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 generalisation function) and then, according to Definition 5.3.4, we prove that $\Omega(I(c))$ is not smaller than k.

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 generalisation tree times the complexity of computing Ω.

Before analysing the complexity of *BottomUp*, we first introduce a simple but effective optimisation. In principle the computation of *BottomUpInversion*(c), as described in Procedure 12, 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 optimisation, we never compute $\Omega(c)$ twice for the same ST-cell c. Consequently, in the worst-case (i.e., when *BottomUp* traverses the generalisation tree up to the root), we need to compute Ω for each node in the tree.

Comparing the two algorithms, given a generalisation tree of height h, *TopDown* requires computing Ω a number of times linear in h, since *TopDown* "moves down" in the generalisation tree at each iteration. Vice versa, in the worst case *BottomUp* needs to process all nodes of the generalisation 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 generalisation tree (see Section 5.3.3), 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 generalisation trees we get smaller bottom ST-cell and this positively impacts on the average size of the generalisation 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 5.3.4 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 5.3.4, 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 generalisation 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 generalisation 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*.

5.3.4 Experimental evaluation

In this section we evaluate the SafeBox computation algorithms described in Section 5.3.3. Using two different datasets, we first evaluate the effectiveness of *TopDown* and *BottomUp* algorithms by showing how parameter k impacts on the data and service 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* [41].

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 5.3.4, while the second dataset includes all the shops in Milano with opening hours (see Section 5.3.4). 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 5.3.3, 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 generalisation 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 5.3.3, 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 5.1 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 5.1: Spatial and temporal leaf ST-cell size

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[4].

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 5.3.2. In Table 5.2 the values of k are summarised.

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

Table 5.2: MilanoByNight k values

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

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, that we consider a trusted third party in our architecture. 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 5.3.2: 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 5.3.

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

Table 5.3: GoogleShops parameters values

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 optimisations are possible depending on the application context. 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 generalisation 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 centre. 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 generalisation 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 5.3.4 and 5.3.4), 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.

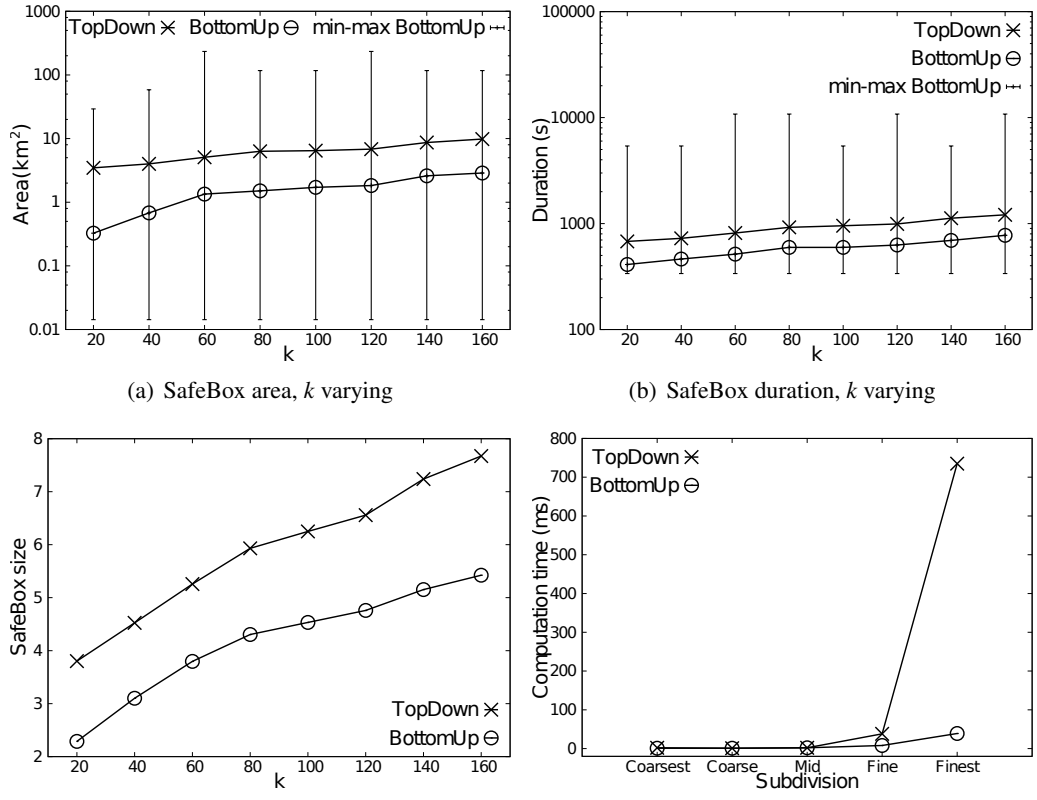
Evaluation

In this section we analyze and discuss the experimental results. In Section 5.3.4 we present the MilanoByNight results, and in Section 5.3.4 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 5.3.4. 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 5.3.4 we compare the *TopDown* and *BottomUp* with the *Grid* algorithm [41].

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 5.3.3).

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 5.12: Results with MilanoByNight dataset

In Figure 5.12(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, hence granting a better service quality. A similar result is shown in Figure 5.12(b), in which the duration of the SafeBoxes is compared.

In Figure 5.12(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 generalisation 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 generalisations.

The overall precision, in both time and space, is summarised in Figure 5.12(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 generalisation tree described in Section 5.3.3. This implies that lower is the value, the smaller is the SafeBox in both time and space, and hence the higher is the precision and the service quality. Figure 5.12(c) confirms the experimental results of area and duration comparisons: the *BottomUp* algorithm produces smaller SafeBoxes up to 3 levels in the generalisation tree, meaning that a *TopDown* SafeBox can be $2^3 = 8$ times bigger than a *BottomUp* SafeBox.

Figure 5.12(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.

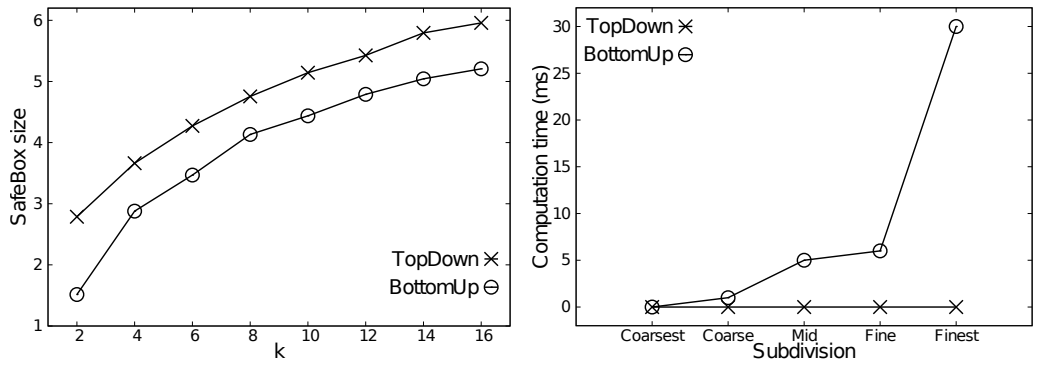
Evaluation with Google shop dataset with appearance semantics

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

Figure 5.13(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 5.13(a). Therefore, we can conclude that the *BottomUp* algorithm is preferable since it produces on average smaller SafeBox, granting a better data and service quality.

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 5.13(b): we can observe an opposite trend with respect to the MilanoByNight dataset. The computation time of *TopDown* is always less than



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

Figure 5.13: Results with Google Shops dataset - Appearance semantics

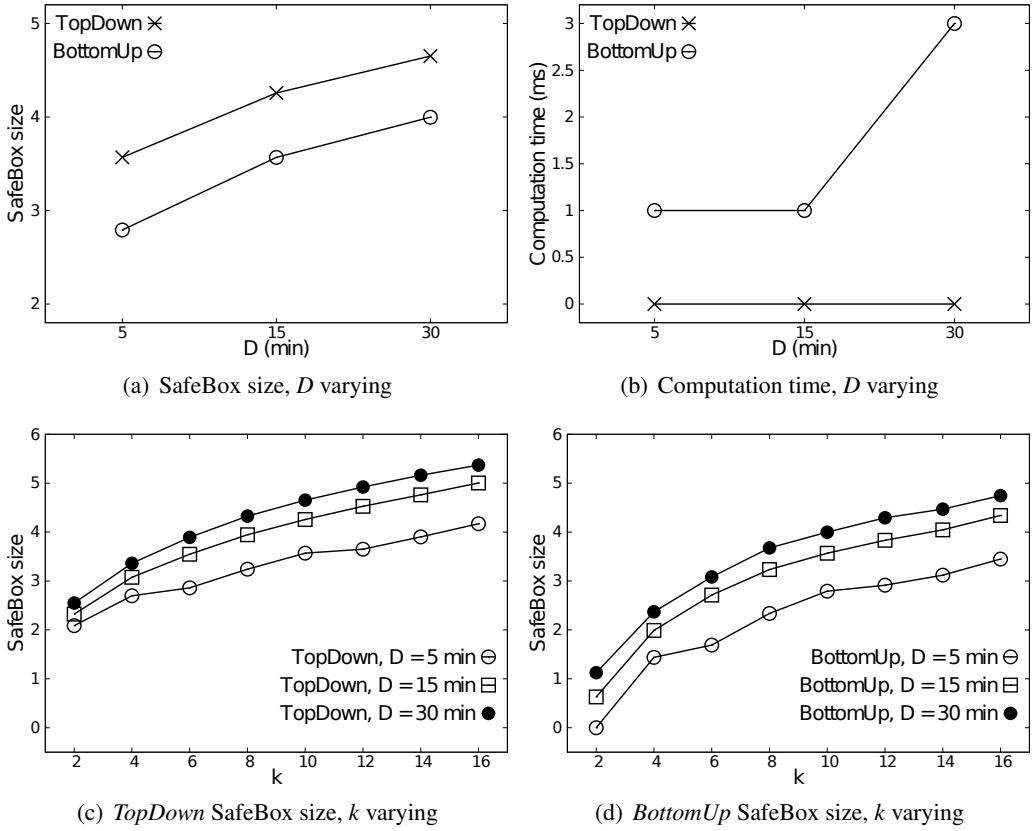
one millisecond while the execution time of the *BottomUp* algorithm is slightly affected by the increase of the number of leaf ST-cells. Indeed we can observe that with the finest subdivision *BottomUp* takes up to 30 milliseconds. The difference with respect to the MilanoByNight dataset is due to the 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.

Evaluation with Google shop dataset with persistence semantics

This set of experiments evaluates the two algorithms precision and performance with the Google shops dataset adopting the persistence semantics. Figure 5.14(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 Safe10000Boxes.

Figure 5.14(b) shows the computation time of the algorithms varying the persistence interval duration. As also shown in Figure 5.13(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. Indeed, the *Residuals* function is on average called 120 times with persistence interval duration $D = 5$ minutes and nearly 800 times considering a persistence interval duration of 30 minutes.

Figure 5.14(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 5.13(a)



(a) SafeBox size, D varying

(b) Computation time, D varying

(c) *TopDown* SafeBox size, k varying

(d) *BottomUp* SafeBox size, k varying

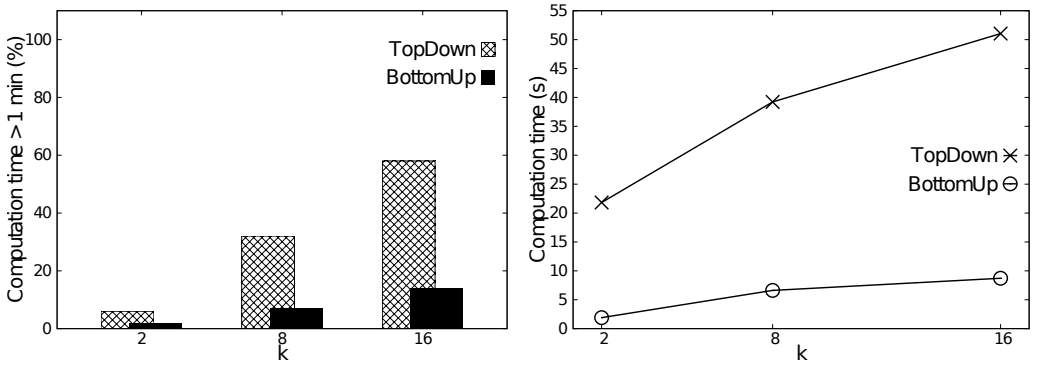
Figure 5.14: Results with Google Shops dataset - Persistence semantics

and in Figure 5.14(a). Figure 5.14(d) shows a similar result for *BottomUp*.

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

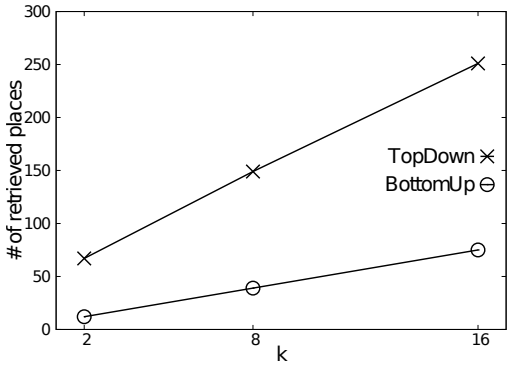
Similarly to Section 5.3.4, in this set of experiments we evaluate the performance of *TopDown* and *BottomUp* algorithms with the Google Shops dataset adopting the appearance semantics and varying k. In this set of experiments when data is needed to compute Ω we retrieve it from Google servers through the Google Places API. Clearly, in this set of experiments the generalisation 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 generalisation. If the generalisation algorithm does not terminate within this time, the generalisation procedure is interrupted and it is considered "timed-out".

Figure 5.15(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 5.15(c)). However, while with *TopDown* more than 60% of the generalisation runs are timed out with $k = 16$, with *BottomUp* the percentage is much



(a) Percentage of failed experiments, k varying

(b) Computation time, k varying



(c) Number of Places, k varying

Figure 5.15: Results with online retrieval of Google Shops dataset

smaller (i.e., less than 10%).

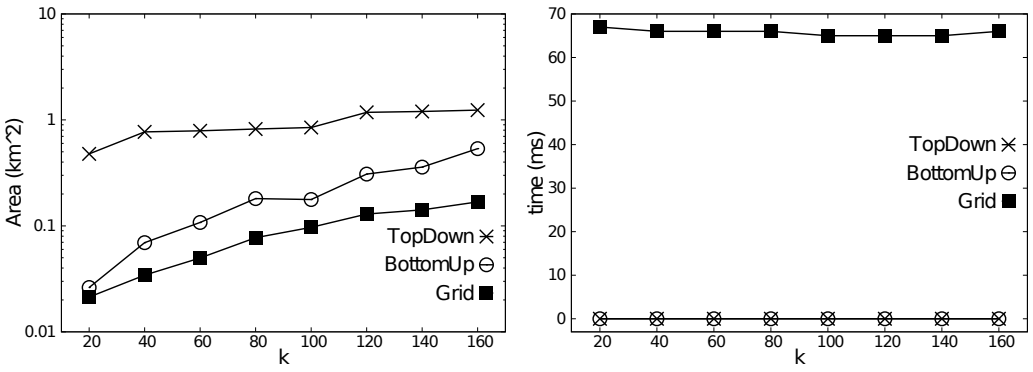
Figure 5.15(b) shows the average computation time computed among the runs that were not timed-out. We can observe that *TopDown* has a much larger computation time than *BottomUp* for different values of k. The main reason for this result is that *BottomUp* requires to retrieve less points. Indeed in Figure 5.15(c) we can observe that, for all considered values of k, *BottomUp* requires about one fifth of the points that are needed by *TopDown*.

Comparison with Grid algorithm

In this last set of experiments we compare the *TopDown* and *BottomUp* algorithms with the *Grid* spatial generalisation algorithm, presented in [41].

Grid

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 generalised area. This algorithm partitions the set of objects in two steps. During the first step, the objects are totally ordered considering their location along the x axis, then along the y axis and



(a) SafeBox size, k varying

(b) Computation time, k varying

Figure 5.16: Comparison with existing solutions

eventually according to their user identifier. This ordered set of objects is then partitioned into blocks of consecutive objects, each block having the same number of objects except the last block that may contain more objects than the previous ones. Only the objects that are in the same block as the issuer, are considered in the second step in which objects are ordered considering first their location along the y axis, then their location along the x axis and eventually according to their user identifier. Using this ordering, all the objects are partitioned similarly to what is done in the first step. Eventually, the set of objects that are in the same block as the issuer is returned.

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 generalisation.

In Figure 5.16(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 5.16(b) shows that *Grid* has a much higher computation time.

In our previous work [41] we compared *Grid* with the *Hilbert Cloak* algorithm presented in [29]. 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.

5.4 Summary

In this section we addressed the problem of privacy preservation through spatial and spatio-temporal generalisation and we described the formal characteristics of three generalisation

technique specifically designed for privacy-aware systems.

We first formalise $\mathscr{G}$onio and $\mathscr{A}$equus spatial generalisations and we study their computational characteristics. In particular our $\mathscr{A}$equus granules have the property that, independently from the latitude considered, each region at the same level has the same area, characteristic that is not granted in $\mathscr{G}$onio generalisation and in others similar proposal in the literature. We formally prove that the basic operations of both $\mathscr{G}$onio and $\mathscr{A}$equus granularities can be computed in constant time.

With the SafeBox proposal we presented a new problem formalisation that makes the generalisation technique independent from the semantics of the generalised region. This approach makes it possible to define generalisation 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 generalised 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 generalisation 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 generalisation 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.

6. Conclusions

In this dissertation we have focused on the risks for the user's privacy when the location information is shared with other users and with the service provider.

6.1 Summary of the contributions

The first goal was to highlights the danger in indiscriminately sharing locations by the users. We have studied the practical threats in using distance preserving transformations as privacy preserving technique in the context of friend finder services. Even if distance preserving transformations have already been proven as theoretically unsafe, we studied them practically and we have discovered that is feasible to infer users' locations quite precisely. After studying comprehensively an already known threat, we have faced the formalisation of a "new threat": the co-location privacy issue in Geo-Social Networks. We believe that the formalisation of this threat is the foundation upon which a targeted privacy preserving technique can be developed.

Once the risks for the users' privacy have been spotlighted in the first part, the second objective of this dissertation was to protect users' location privacy. We have focused on two main categories:

1. For privacy techniques that make use of space transformations we have proposed a novel technique for distance-based services capable of protecting users' privacy even if the adversary knows 1) the users' distribution in the space 2) the distances between some pairs of users. The safety of distance preserving transformations is forced through a uniformation of the users' distribution in the space. We have proposed two novel algorithms capable of computing such uniformation and we have experimentally evaluated them.

2. For privacy techniques in which the privacy was granted by the use of a location's generalisation, we have proposed three (spatial and/or temporal) cloaking techniques. These

generalisation techniques are tools, specifically designed for location privacy, that can be used by any techniques in which a location generalisation is required. The generalisations are designed to be fast to compute and flexible enough to adapt to several cases of use, in order to fit in many privacy preserving techniques.

In this dissertation, through the identification of the threats and the proposal of new privacy preserving techniques and tools, we have contributed to build the basis upon which the protection of the users' location privacy will be, one day, preserved in every context and application.

6.2 Future works and general location privacy issues

Undoubtedly, we are far from a comprehensive solution that can solve the overall location privacy issues.

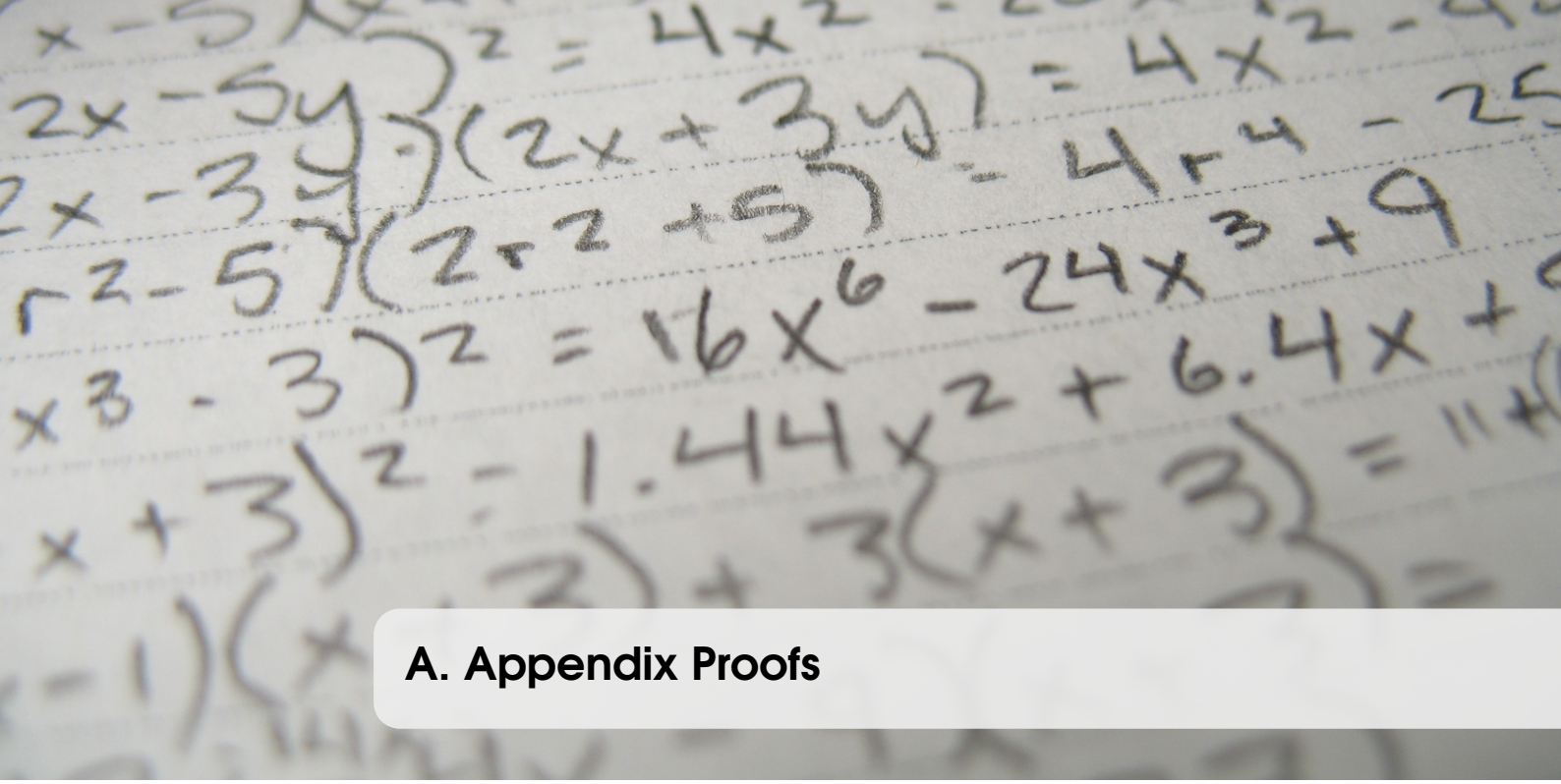
First of all, new threats are stemming from the exploitations of the users' locations in new applications and services. As the birth and the spread of Geo-Social Networks have caused new risks for the users' privacy (as absence and co-location), it is possible that in the future new location-based applications will create the conditions under which novel privacy threats will spawn. Consequently, a future work will be the formalisation of these new privacy threats, since we believe that such formalisation is needed to better understand the problem, before proposing new privacy preserving techniques.

Even without looking to new privacy threats and despite the irrefutable efforts of the researchers that have proposed techniques for protecting the location privacy so far, nowadays the users' location privacy is not always protected and new privacy preserving techniques are needed. We believe that each proposed privacy protection technique should 1) protect the users' privacy efficiently 2) be compliant with the business model. Since most of the location based services and geo-social networks are business-oriented it is necessary that the privacy techniques could be usable in this context, that it is often conflicting with the academic one. An interesting future work will be the proposal of an effective solution that could protect the users' against the co-location threat in Geo-Social Networks, that we have formalised in Chapter 3. Moreover, since we have seen in Chapter 5 how to compute a spatio-temporal generalisation given a location and a time, an extension can be the computation of such spatio-temporal taking into account trajectories instead of precise positions.

A question that often arises when taking into consideration a privacy preserving technique is: how much this technique protects the users? The quantification of location privacy is not straightforward and can be challenging to compare different techniques, that are targeted to protect users' privacy from adversaries with specific knowledge. As a future work, we want to investigate in this direction, such that it could be easier to understand the level of protection granted by a given technique, evaluating its effectiveness also in real settings.

Another problem is to understand how the users perceive the threats in sharing and using their locations, or in other words, how can we, as scientists, capture users' privacy preferences? It is not feasible that the users themselves decide all the settings that are often needed for a privacy preserving technique to work properly. We believe that much work has done in this direction but further it will be needed in the future to extract and infer correctly users' privacy preferences in a intuitive way.

Concluding, we have seen that location privacy is a complex topic, whose complexity derives from the intrinsic quantity of information that resides in the location itself, as discussed in Chapter 1. This complexity reflects in the difficulties of finding definitive solutions to protect the users' privacy. For this reason we believe that a mere technical solution, no matter how sophisticated, cannot be sufficient without the assistance of laws. If there is no limits upon the quantity and the precision of the information that a company or other persons can legally acquire on the users, then there will be not so much to do from a technical point of view. For these reasons, in the future, we wish for a strict and useful collaboration between institutions' legislators and computer scientists, such that users' privacy will be eventually preserved under any circumstances.



A. Appendix Proofs

A.1 Proof of Theorem 3.2.1

Proof. Let's consider the area $iPlus_D$: if the user u is located everywhere else then the closeness probability is 0, since the two users locations are farther than D. Indeed, if we consider

$$diffC_u = c_u - iPlus_D$$

then $\forall p \in diffC_u$ and $\forall p' \in c_{u'}\ d(p,p') > D$. Given that $p = loc_u(t)$ with $p \in iPlus_D$, we now want to compute the probability $p_{u'}$ that the user u' is within distance D from p. This probability is equal to check the probability that u' is located in the circle c^p_{iD} centred in p with radius D, precisely

$$p_{u'} = \frac{A(c^p_{iD} \cap c_{u'})}{A(c_{u'})}$$

The intersection between c^p_{iD} and $c_{u'}$ is needed to compute with precision the probability of u' to be within distance D from p, because the probability that u' is located in p is null for each $p \notin c_{u'}$. From set theory we know that $A \supset (A \cap B)$. If A and B are spatial regions, we can conclude that the area of the region A is greater or equal than the area of the intersection between A and the region B. Since $A(c^p_{iD}) \geq A(c^p_{iD} \cap c_{u'})$, $A(c^p_{iD})$ is an upper bound for $A(c^p_{iD} \cap c_{u'})$. Hence, we approximate the probability $p_{u'}$ with the area of a circle of area D, c^p_{iD}. Furthermore since the area of c^p_{iD} does not depend on its center, we can compute $c^p_{iD} = \pi \cdot D^2$

$$p_{u'} = \frac{A(c^p_{iD} \cap c_{u'})}{A(c_{u'})} \approx pApprox_{u'} = \frac{A(c^p_{iD})}{A(c_{u'})} = \frac{\pi \cdot D^2}{A(c_{u'})}$$

we can conclude that

$$p_{u'} \leq pApprox_{u'}$$

In conclusion,

$$Pclose_D(t,u,u') \leq \frac{A(iPlus_D)}{A(c_u)} \cdot pApprox_{u'} = \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}$$

■

A.2 Proof of Theorem 3.2.2

Proof. Let's consider the circle $cMinus_D$ centred in $(x_{u'}, y_{u'})$ with radius $(r' - D)$: $iMinus_D = cMinus_D \cap c_u$. $iMinus_D$ is an approximation of the area in which u can be located to have closeness probability $\neq 0$

For construction, for all the points $p \in cMinus_D$, and (since $iMinus \subset cMinus_D$ this holds for all the points $p \in iMinus_D$) the circle centred in p with radius D stand inside $c_{u'}$. If the user u is located in a point $p \in iMinus_D \subset cMinus_D$ then the probability that u' is located in the circle centred in p with radius D is always equal to $\frac{A(c^p_{iD})}{A(c_{u'})}$. Given that $p = loc_u(t)$ with $p \in iMinus_D$, we now want to compute the probability $p_{u'}$ that the user u' is within distance D from p. The exact closeness probability is equal to

$$Pclose_D(t,u,u') = \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{A(c_{iD})}{A(c_{u'})}$$

Since $iPlus_D \supset (c_u \cap c_{u'}) \supset iMinus_D$ and consequently $A(iPlus) > A(iMinus)$, it holds that:

$$Pclose_D(t,u,u') = \frac{A(iPlus_D)}{A(c_u)} \cdot \frac{A(c_{iD} \cap c_{u'})}{A(c_{u'})} \geq \frac{A(iMinus_D)}{A(c_u)} \cdot \frac{A(c_{iD} \cap c_{u'})}{A(c_{u'})}$$

where for all points $p \in iMinus_D$ $A(c^p_{iD} \cap c_{u'}) = A(c^p_{iD}) = \pi \cdot D^2$. Hence,

$$Pclose_D(t,u,u') \geq \frac{A(iMinus_D)}{A(c_u} \cdot \frac{A(c_{iD})}{A(c_{u'})} = \frac{A(iMinus_D)}{A(c_u} \cdot \frac{\pi \cdot D^2}{A(c_{u'})}$$

■

A.3 Proof of Theorem 4.3.1

Proof. a) Property of the transformation. We have to show that

$$\int_0^x \int_0^y g(w,z)dw\,dz = \frac{x' \cdot y'}{|S|}$$

where $(x', y') = f(x,y)$ and $f()$ is the function computed by the algorithm. The proof has 2 steps: a1) the property holds considering separately each cell c in S and its transformation in c' by the algorithm. This can be verified by considering as point $(0,0)$ the lower left corner of the cell and as point (x,y) any other point in the cell. It is easily seen that

$$\frac{Area(c)}{|S'|} = \frac{NumU(c)}{NumU(S)}$$

i.e., the area of the deformed space corresponding to c with respect to the total area of the space has the same proportion than the number of users in the original c with respect to the total number of users.
a2) The second step shows that if the property holds for each cell in S, then it holds for the whole space S. Intuitively, any rectangular area R in S' identified with its lower left corner in $(0,0)$ and upper right corner in (x',y') with $x' \leq x_{max}$ and $y' \leq y_{max}$ can be partitioned in rectangles that are properly contained in a cell c'_i (i.e., in one of the deformed areas). In some cases the rectangle will have as lower left corner the lower left corner of the cell, in others will be the complementary space with respect to one that has the lower left corner of the cell. It is easily seen that the property holds for each of the sub-rectangles. Then, if R_1, R_2 are two of these rectangles, observing that they do not overlap, we have that

$$\frac{Area(R_1)}{|S'|} = \frac{NumU(R_1)}{NumU(S)} \text{ and } \frac{Area(R_2)}{|S'|} = \frac{NumU(R_2)}{NumU(S)}$$

implies

$$\frac{Area(R_1) \cup Area(R_2)}{|S'|} = \frac{NumU(R_1) + NumU(R_2)}{NumU(S)}$$

Since this can be applied to the whole set of sub-rectangles partitioning R, the property holds for R.

■

A.4 Proof of Theorem 4.3.2

Proof. Let $Z(mxm)$ be the number of steps the uniformRecursively algorithm takes on an (mxm) grid. Assume $(mxm) = n$ so $m = sqrt(n)$. Now we have the following recurrence relationship based on the algorithm:
$Z(mxm) = Z(k1xk2) + Z(k3xk4) + 2m$, where $k1 + k2 = m$
$Z(1x1) = 1$.

So the algorithm takes $2m$ time to decide where (and type) to cut, and only takes constant time to do the real "cutting". And then calls itself on the two partitions of sizes $(k1xk2)$ and $(k3xk4)$. Note either $k1$ or $k2$ is a multiple of m, and either $k3$ or $k4$ is a multiple of m due to how we cut.) So the execution is like a tree. The worst scenario (most steps for Z) is totally unbalanced tree, i.e., $k1 = 1$ and $k2 = m$, in which case $k3xk4 = (m-1)xm$.

Clearly, $Z(1xm) = m$. Hence worst case: $Z(mxm) = Z(m-1xm) + 3m$. Now it should be clear the highest tree that $Z(mxm)$ generates is of height at most $2m$.
$Z(mxm) = Z(m-1xm) + 3m$
$Z(m-1xm) = Z(m-2xm) + 3(m-1)$
...
$Z(1xm) = m+3$
(i.e., each step uses the worst case scenario and the above is about $2m$ lines.). So $Z(mxm) = 3m + 3(m-1) + \ldots + 3*0 + 3 + m$ (this has at most $2m$ items)
Hence $Z(mxm) = O(mxm) = O(n)$. ■

A.5 Proof of Property 5.2.1

Proof. By Definition 5.2.1, we need to prove that index $i = \mathscr{G}^l[p]$ is defined, for any $l \in \mathbb{N}$ and for any $p = \langle lat, lon \rangle \in S$. Since $lon \in [-180, 180)$, by Equation 5.6, $X_i \in [0, 2^l) = [0, 2^l - 1]$. Since $lat \in (-90, 90)$, by Equation 5.7, $Y_i \in [0, 2^l) = [0, 2^l - 1]$. Consequently, by Equation 5.1, $i \in [0, 4^l)$. By Definition 5.2.3, $\mathscr{G}^l(i)$ is defined. ■

A.6 Proof of Property 5.2.2

Proof. By Definition 5.2.2, we need to prove that, for any level l, given $i, j \in [0, 4^l)$, with $i \neq j$, $\mathscr{G}^l(i) \cap \mathscr{G}^l(j) = \emptyset$. Since $i \neq j$, then $X_i \neq X_j$ or $Y_i \neq Y_j$ or both. We now show that if $X_i \neq X_j$ then $\mathscr{G}^l(i) \cap \mathscr{G}^l(j) = \emptyset$. The proof is analogous for the case $Y_i \neq Y_j$. Without loss of generality, let $X_i < X_j$. By Definition 5.2.3, *maxLon* of $\mathscr{G}^l(i)$ is less than or equal to *minLon* of $\mathscr{G}^l(j)$ and hence, by Definition 5.2.3, $\mathscr{G}^l(i) \cap \mathscr{G}^l(j) = \emptyset$.

■

A.7 Proof of Property 5.3.1

Proof. By Definition 5.3.3, for any spatio-temporal area A, $I(A) \leq A$. Hence, $I(\mathscr{G}(p)) \leq \mathscr{G}(p)$. Consequently, due to Definition 5.3.1, $\Omega(\mathscr{G}(p)) \geq \Omega(I(\mathscr{G}(p)))$. Since, by Definition 5.3.4, for any source point p such that $\mathscr{G}(p)$ is defined, $\Omega(I(\mathscr{G}(p))) \geq k$, then $\Omega(\mathscr{G}(p)) \geq \Omega(I(\mathscr{G}(p))) \geq k$. Hence, by Definition 5.3.2, $\mathscr{G}(p)$ is a *SafeBox*. ■

A.8 Proof of Property 5.3.2

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 5.3.1 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 5.3.6, $\Omega(A) \leq \Omega(A')$. ■

A.9 Proof of Property 5.3.3

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 5.3.1 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 5.3.7, $\Omega(A) \leq \Omega(A')$. ■

A.10 Proof of Theorem 5.3.4

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 5.3.4. 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 5.3.3, 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.11 Proof of Theorem 5.3.5

Proof. By definition of *TopDownBasic* and *TopDown*(Algorithms 9 and 10) 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 9 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.12 Proof of Theorem 5.3.7

Proof of Theorem 5.3.7 easily follows from the proof of Property 5.3.6 and is detailed in Section A.12.2.

A.12.1 Proof of Property 5.3.6

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 5.3.3, 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

$\Omega(\mathit{BottomUpInversion}(c)) \geq k$. Consequently, by Definition 5.3.3, $I(c) = \mathit{res}(c)$. We now show that, for each ST-cell c, $\mathit{Residuals}(c) = \mathit{res}(c)$. 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*, $\mathit{res}(c) = c$. By definition of *Residuals*, $\mathit{Residuals}(c) = c$.

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

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

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

Overall:

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

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

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

The thesis follows since, by induction assumption, $\mathit{Residuals}(c_1) = \mathit{res}(c_1))$ and, analogously, $\mathit{Residuals}(c_2) = \mathit{res}(c_2))$.

■

A.12.2 Proof of Theorem 5.3.7

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 belongs to $\mathit{BottomUp}(p, \alpha, \Omega(), k)$, if $\mathit{BottomUp}(p, \alpha, \Omega(), k)$ does not return **fail**. In Line 1 of the algorithm, variable $\mathit{currentSTcell}$ is set to the leaf ST cell that contains p, hence, clearly $p \in \mathit{currentSTcell}$.

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 5.3.4. 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 $\Omega(BottomUpInversion(c)) \geq k$. Since $BottomUpInversion(c) = I(c)$ (by Property 5.3.6), it holds that $\Omega(I(c)) \geq k$. ■

Bibliography

[1] Osman Abul, Francesco Bonchi, and Mirco Nanni. “Never Walk Alone: Uncertainty for Anonymity in Moving Objects Databases”. In: *Proc. of the 24th Int. Conf. on Data Engineering*. IEEE, 2008 (cited on page 83).

[2] Claudio Agostino Ardagna, Marco Cremonini, Ernesto Damiani, S De Capitani Di Vimercati, and Pierangela Samarati. “Location privacy protection through obfuscation-based techniques”. In: *Data and Applications Security XXI*. Springer, 2007, pages 47–60 (cited on pages 15, 83).

[3] Bhuvan Bamba, Ling Liu, Peter Pesti, and Ting Wang. “Supporting anonymous location queries in mobile environments with privacygrid”. In: *Proc. of the 17th Int. Conf. on World Wide Web*. ACM. 2008, pages 237–246 (cited on pages 14, 15, 83).

[4] Letizia Bertolaja. “Location Privacy Management and Protection in Geo-Social Networks”. In: *Proc. of the 14th International Conference on Mobile Data Management (MDM 2013)*. Ph.D. forum paper. IEEE Comp. Soc. 2013.

[5] Letizia Bertolaja, Dragan Ahmetovic, and Sergio 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 (cited on pages 10, 82, 83, 102).

[6] Claudio Bettini, Sushil Jajodia, X Sean Wang, and Pierangela Samarati. *Privacy in Location-Based Applications*. Springer, 2009 (cited on pages 13, 20, 81).

[7] Thomas Bittner and Barry Smith. “A Taxonomy of Granular Partitions”. English. In: *Spatial Information Theory*. L.N.C.S. Springer, 2001 (cited on page 84).

[8] Chi-Yin Chow, Mohamed F Mokbel, and Walid G Aref. “Casper*: Query processing for location services without compromising privacy”. In: *ACM Transactions on Database Systems (TODS)* 34.4 (2009), page 24 (cited on page 14).

[9] Landon P Cox, Angela Dalton, and Varun Marupadi. "Smokescreen: flexible privacy controls for presence-sharing". In: *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM. 2007, pages 233–245.

[10] Maria Luisa Damiani, Elisa Bertino, and Claudio Silvestri. "The PROBE Framework for the Personalized Cloaking of Private Locations." In: *Transactions on Data Privacy* 3.2 (2010), pages 123–148 (cited on pages 82, 84, 85).

[11] Iginia De Fent, Donatella Gubiani, and Angelo Montanari. "Granular GeoGraph: a multi-granular conceptual model for spatial data". In: *Symposium on Advanced Database Systems*. 2005 (cited on page 84).

[12] Aristide Fattori, Alessandro Reina, Andrea Gerino, and Sergio Mascetti. "On the Privacy of Real-World Friend-Finder Services". In: *Proc. of the 14th Int. Conf. on Mobile Data Management*. IEEE, 2013.

[13] Thore Fechner and Christian Kray. "Attacking location privacy: exploring human strategies". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp. ACM, 2012, pages 95–98 (cited on page 8).

[14] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. "Preserving location and absence privacy in geo-social networks". In: *Proceedings of the 19th ACM international conference on Information and knowledge management*. CIKM '10. New York, NY, USA: ACM, 2010, pages 309–318 (cited on pages 16, 38).

[15] Gerald Friedland and Robin Sommer. "Cybercasing the joint: on the privacy implications of geo-tagging". In: *Proceedings of the 5th USENIX conference on Hot topics in security*. HotSec'10. USENIX Association, 2010, pages 1–8 (cited on page 16).

[16] Michael T. Gastner and M. E. J. Newman. "Diffusion-based method for producing density-equalizing maps". In: *Proceedings of the National Academy of Sciences of the United States of America* 101.20 (2004), pages 7499–7504 (cited on pages 58, 59).

[17] Bugra Gedik and Ling Liu. "Location Privacy in Mobile Systems: A Personalized Anonymization Model". In: *Proc. of 25th ICDCS Int. Conf. on Distributed Computing Systems*. 2005.

[18] Bugra Gedik and Ling Liu. "Protecting location privacy with personalized k-anonymity: Architecture and algorithms". In: *Mobile Computing, IEEE Transactions on* 7.1 (2008), pages 1–18 (cited on pages 9, 14, 82, 83).

[19] Gabriel Ghinita. "Private queries and trajectory anonymization: a dual perspective on location privacy". In: *Tran. on Data Privacy* (2009) (cited on pages 20, 84).

[20] Gabriel Ghinita, Maria Luisa Damiani, Claudio Silvestri, and Elisa 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 (cited on pages 16, 83, 94).

[21] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. "Private queries in location based services: anonymizers are not necessary". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pages 121–132 (cited on page 16).

[22] Gabriel Ghinita, Carmen Ruiz Vicente, Ning Shang, and Elisa Bertino. "Privacy-preserving matching of spatial datasets with protection against background knowledge". In: *Proc. of the 18th SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*. San Jose, California: ACM, 2010 (cited on pages 21, 37, 38).

[23] Philippe Golle and Kurt Partridge. "On the Anonymity of Home/Work Location Pairs". In: *Proceedings of the 7th International Conference on Pervasive Computing*. Springer-Verlag, 2009, pages 390–397 (cited on page 8).

[24] Herbert L. Groginsky. "Position Estimation Using Only Multiple Simultaneous Range Measurements". In: *IRE Transactions on Aeronautical and Navigational Electronics* (1959) (cited on page 30).

[25] Marco Gruteser and Dirk 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 (cited on pages 9, 14, 15, 82).

[26] Baik Hoh, Marco Gruteser, Hui Xiong, and Ansaf Alrabady. "Preserving privacy in GPS traces via uncertainty-aware path cloaking". In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pages 161–171 (cited on page 16).

[27] Jan Holvast. *History of privacy*. Springer, 2009 (cited on page 9).

[28] Haibo Hu and Jianliang Xu. "Non-Exposure Location Anonymity". In: *Proc. of the 25th Int. Conf. on Data Engineering*. IEEE, 2009 (cited on page 20).

[29] Panos Kalnis, Gabriel Ghinita, Kyriakos Mouratidis, and Dimitris Papadias. "Preventing location-based identity inference in anonymous spatial queries". In: *Transactions on Knowledge and Data Engineering* 19.12 (2007), pages 1719–1733 (cited on pages 9, 14, 82, 83, 91, 95, 116).

[30] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. "An anonymous communication technique using dummies for location-based services". In: *Proceedings of the International Conference on Pervasive Services*. IEEE Computer Society, 2005 (cited on page 14).

[31] Jemima Kiss. "Facebook: Did anyone really quit?" In: *The Guardian*. London, 2010. URL: `"http://www.guardian.co.uk/media/pda/2010/jun/01/digital-media-facebook"` (cited on page 9).

[32] Hong Ping Li, Haibo Hu, and Jianliang Xu. "Nearby Friend Alert: Location Anonymity in Mobile Geo-Social Networks". In: *IEEE Pervasive Computing* (2012). ISSN: 1536-1268 (cited on pages 16, 20).

[33] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity." In: *ICDE*. Volume 7. 2007, pages 106–115 (cited on page 14).

[34] Kun Liu, Chris Giannella, and Hillol Kargupta. "An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining". In: *Proc. of the 10th Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases*. Springer, 2006 (cited on pages 10, 20, 53).

[35] Zheli Liu, Jin Li, Xiaofeng Chen, Jingwei Li, and Chunfu Jia. "New privacy-preserving location sharing system for mobile online social networks". In: *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*. IEEE. 2013, pages 214–218 (cited on page 16).

[36] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. "l-diversity: Privacy beyond k-anonymity". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), page 3 (cited on page 14).

[37] Sergio Mascetti, Letizia Bertolaja, and Claudio Bettini. "Location privacy attacks based on distance and density information". In: *Proc. of the 20th SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*. ACM, 2012 (cited on pages 10, 19, 21).

[38] Sergio Mascetti, Letizia Bertolaja, and Claudio Bettini. "A Practical Location Privacy Attack in Proximity Services". In: *Proc. of the 14th International Conference on Mobile Data Management (MDM 2013)*. Full paper. IEEE Comp. Soc. 2013 (cited on pages 10, 19).

[39] Sergio Mascetti, Letizia Bertolaja, and Claudio Bettini. "SafeBox: adaptable spatio-temporal generalization for location privacy protection". In: *Transactions on Data Privacy* 7.2 (2014). Journal paper, pages 131–163 (cited on pages 10, 82).

[40] Sergio Mascetti, Claudio Bettini, and Dario Freni. "Longitude: Centralized Privacy-Preserving Computation of Users' Proximity". In: *Proc. of 6th VLDB workshop on Secure Data Management*. LNCS. Springer, 2009 (cited on pages 10, 14, 20, 54, 72).

[41] Sergio Mascetti, Claudio Bettini, Dario Freni, and X. Sean Wang. "Spatial Generalization Algorithms for LBS Privacy Preservation". In: *Journal of Location Based Services* 2.1 (Mar. 2008) (cited on pages 14, 82, 83, 91, 95, 108, 111, 115, 116).

[42] Sergio Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, and Sushil Jajodia. "Privacy-Aware Proximity Based Services". In: *Proc. of the 10th Int. Conf. on Mobile Data Management*. MDM. IEEE, 2009 (cited on page 83).

[43] Sergio Mascetti, Claudio Bettini, X. Sean Wang, Dario Freni, and Sushil Jajodia. "ProvidentHider: an algorithm to preserve historical k-anonymity in LBS". In: *In Proc. of the 10th International Conference on Mobile Data Management*. IEEE Computer Society, 2009 (cited on pages 9, 14, 83, 94).

[44] Sergio Mascetti, Dario Freni, Claudio Bettini, X.Sean Wang, and Sushil Jajodia. "Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies". English. In: *The VLDB Journal* 20.4 (2011), pages 541–566 (cited on pages 13, 16, 20, 54, 73, 80, 83).

[45] Sergio Mascetti, Anna Monreale, Annarita Ricci, and Andrea Gerino. “Anonymity: A Comparison Between the Legal and Computer Science Perspectives”. In: *European Data Protection: Coming of Age*. Springer, 2013, pages 85–115 (cited on page 10).

[46] Mohamed F Mokbel, Chi-Yin Chow, and Walid 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 (cited on page 83).

[47] Anna Monreale, Gennady Andrienko, Natalia Andrienko, Fosca Giannotti, Dino Pedreschi, Salvatore Rinzivillo, and Stefan Wrobel. “Movement data anonymity through generalization”. In: *Transactions on Data Privacy* (2010).

[48] Paul Ohm. “Broken promises of privacy: Responding to the surprising failure of anonymization”. In: *UCLA L. Rev.* 57 (2009), page 1701 (cited on page 10).

[49] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. “Efficient OLAP Operations in Spatial Data Warehouses”. In: *Proc. of the 7th SSTD Int. Symposium on Advances in Spatial and Temporal Databases*. Springer-Verlag, 2001 (cited on page 84).

[50] Tatiana Pontes, Gabriel Magno, Marisa Vasconcelos, Aditi Gupta, Jussara Almeida, Ponnurangam Kumaraguru, and Virgilio Almeida. “Beware of what you share: Inferring home location in social networks”. In: *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*. IEEE. 2012, pages 571–578 (cited on page 16).

[51] Carmen Ruiz Vicente, Dario Freni, Claudio Bettini, and Christian S. Jensen. “Location-Related Privacy in Geo-Social Networks”. In: *IEEE Internet Computing* 15.3 (2011), pages 20–27 (cited on page 16).

[52] Peter Ruppel, Georg Treu, Axel Kupper, and Claudia Linnhoff-Popien. “Anonymous user tracking for location-based community services.” In: *Proceedings of theSecond International Workshop on Location and Context-Awareness*. Volume LNCS 3987. Springer, 2006 (cited on page 14).

[53] R. Sibson. “SLINK: an optimally efficient algorithm for the single-link cluster method”. In: *The Computer Journal* (1973) (cited on page 26).

[54] Manolis Terrovitis and Nikos Mamoulis. “Privacy preservation in the publication of trajectories”. In: *Mobile Data Management, 2008. MDM’08. 9th International Conference on*. IEEE. 2008, pages 65–72 (cited on page 16).

[55] Traian Marius Truta and Bindu Vinay. “Privacy Protection: p-Sensitive k-Anonymity Property.” In: *ICDE Workshops*. 2006, page 94 (cited on page 14).

[56] Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen. “A Location Privacy Aware Friend Locator”. In: *Proc. of the 11th Int. Symp. on Spatial and Temporal Databases*. LNCS. Springer, 2009 (cited on pages 15, 20).

[57] Laurynas Šikšnys, Jeppe Rishede Thomsen, Simonas Šaltenis, and Man Lung Yiu. “Private and Flexible Proximity Detection in Mobile Social Networks”. In: *Proc. of the 11th Int. Conf. on Mobile Data Management*. IEEE, 2010 (cited on pages 15, 20, 83).

[58] Marius Wernke, Pavel Skvortsov, Frank Dürr, and Kurt Rothermel. “A classification of location privacy attacks and approaches”. In: *Personal and Ubiquitous Computing* (2012), pages 1–13 (cited on pages 13, 81).

[59] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. “Secure kNN computation on encrypted databases”. In: *Proc. of the 2009 SIGMOD Int. Conf. on Management of data*. Providence, Rhode Island, USA: ACM, 2009 (cited on page 21).

[60] Roman Yarovoy and Francesco Bonchi. “Anonymizing moving objects: how to hide a MOB in a crowd?” In: *Proc. of the 12th Int. Conf. on Extending Database Technology*. ACM, 2009.

[61] Man Lung Yiu, Christian S. Jensen, Xuegang Huang, and Hua Lu. “SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services”. In: *Proceedings 24th International Conference on Data Engineering*. IEEE Computer Society, 2008 (cited on pages 9, 14).

[62] Ge Zhong, Ian Goldberg, and Urs Hengartner. “Louis, lester and pierre: Three protocols for location privacy”. In: *Privacy Enhancing Technologies*. Springer. 2007, pages 62–76 (cited on pages 15, 20, 83).

Other references

[O1] *comScore Reports June 2014 U.S. Smartphone Subscriber Market Share*. 2014. URL: "https://www.comscore.com/Insights/Market-Rankings/comScore-Reports-June-2014-US-Smartphone-Subscriber-Market-Share" (cited on page 7).

[O2] *Directive 95/46/EC of the European Parliament*. 1995. URL: "http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf" (cited on page 9).

[O3] *EU General Data Protection Regulation*. 2012. URL: "http://ec.europa.eu/justice/data-protection/" (cited on pages 9, 10).

[O4] Anshul Gupta, Roberta Cozza, and CK Lu. *Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013*. 2014. URL: "http://www.gartner.com/newsroom/id/2665715" (cited on page 7).

[O5] Columbia University Center for International Earth Science Information Network (CIESIN) and Centro Internacional de Agricultura Tropical (CIAT). *Gridded Population of the World, Version 3 (GPWv3)*. 2005 (cited on pages 29, 33, 75).

[O6] The 93rd Congress of the United States. *The privacy act of 1974*. URL: "http://www.justice.gov/opcl/privstat.htm" (cited on page 9).

[O7] United Nations General Assembly. *The Universal Declaration of Human Rights*. URL: "http://www.un.org/en/documents/udhr/" (cited on page 8).

Acknowledgements

I'd like to thank
my advisor Prof. Claudio Bettini
for all the time spent guiding me during this adventure,
and my co-advisor Sergio Mascetti
for all the help and the advices he gave me: you have been essential.

Lots of love to my parents, my uncles and my entire family
Mom, Dad, Nanda, Mario, Matteo, Chiara, Peter, Gianmarco, Carlotta, Margherita, Leonardo
and Florona.

A big hug to all the people that I have loved and met in these three years:
my colleagues Andrea "big back" Gerino, Matteo Sixtus VI "Z", Berna "the drug dealer",
Zaffar and all the guys of the Everyware Lab.,
my roommate Giugi,
my beloved friends Elisa, Marta, Davide, Lorenzo, Somax and Barbara,
and the Dolce Forno, Piccola Ischia and Torrefazione crews.

At last, a very special thank to Dragan and Christian,
without your support probably I would not have finished my Ph.D. and,
with certainty, without you my life in these three years would have been much more sad.
I love you my friends.