Privacy-Aware Querying over Sensitive Trajectory Data

Nikos Pelekis
Department of Statistics and Insurance Science
University of Piraeus, Greece
npelekis@unipi.gr

Aris Gkoulalas-Divanis
Information Analytics Lab
IBM Research–Zurich
Rüschlikon, Switzerland
agd@zurich.ibm.com

Marios Vodas
Department of Informatics
University of Piraeus
Piraeus, Greece
mvodas@gmail.com

Despina Kopanaki
Department of Informatics
University of Piraeus
Piraeus, Greece
dkopanak@unipi.gr

Yannis Theodoridis
Department of Informatics
University of Piraeus
Piraeus, Greece
ytheod@unipi.gr

ABSTRACT

Existing approaches for privacy-aware mobility data sharing aim at publishing an anonymized version of the mobility dataset, operating under the assumption that most of the information in the original dataset can be disclosed without causing any privacy violations. In this paper, we assume that the majority of the information that exists in the mobility dataset must remain private and the data has to stay in-house to the hosting organization. To facilitate privacy-aware sharing of the mobility data we develop a trajectory query engine that allows subscribed users to gain restricted access to the database to accomplish various analysis tasks. The proposed engine (i) audits queries for trajectory data to block potential attacks to user privacy, (ii) supports range, distance, and k-nearest neighbors spatial and spatiotemporal queries, and (iii) preserves user anonymity in answers to queries by (a) augmenting the real trajectories with a set of carefully crafted, realistic fake trajectories, and (b) ensuring that no user-specific sensitive locations are reported as part of the returned trajectories.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues—*privacy*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Experimentation

Keywords

Privacy-aware query engine, mobility data, anonymity, trajectory database

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

1. INTRODUCTION

The advances in mobile devices, positioning technologies and spatiotemporal database research, have made possible the tracking of mobile devices at a high accuracy, while supporting the efficient storage of mobility data in databases. From this perspective, we have nowadays the means to collect, store and process mobility data of an unprecedented quantity, quality and timeliness. As ubiquitous computing pervades our society, user mobility data represents a very useful but also sensitive source of information. On one hand, the movement traces of the users can aid traffic engineers, city managers and environmentalists towards decision making in a wide spectrum of applications, such as urban planning, traffic engineering and environmental pollution. On the other hand, the disclosure of mobility data to untrusted parties may jeopardize the privacy of the users whose movement is recorded, leading the way to abuse scenarios such as user tailing and profiling. As it becomes evident, the sharing of user mobility data for analysis purposes has to be done only after the data has been protected against potential privacy breaches.

Recently, several methodologies have been proposed to enable privacy-preserving mobility data sharing. Existing approaches, such as [1,7,8,11,17], aim at publishing an anonymous counterpart of the original dataset in which adversaries can no longer match the recorded movement of each user to the real identity of the user. A common assumption that is implicitly made in these approaches is that most of the information that is stored in the original dataset can be disclosed without causing any privacy violations. In this paper, we refrain from this assumption as it can be proven unrealistic in certain data sharing scenarios. Instead, we employ a more conservative approach to privacy by assuming that the majority of the information that is captured in the mobility dataset must remain private and that *the data has to stay in-house to the hosting organization.* Our assumption is primarily based on the following arguments:

- The data owner may be reluctant to publish the entire mobility dataset, or conformance to certain business regulations may require that the dataset resides in-house to the hosting organization.
- Mobility datasets can typically support many types of data analysis. In order for the anonymous dataset to

be useful in practical applications, it is necessary that the anonymization approach can offer specific utility guarantees and this, in turn, requires knowledge of the intended workload. When the data resides in-house, the privacy preservation algorithms can support many types of data analysis (which may be unknown apriori) by guaranteeing at the same time the privacy of the users, whose information is recorded in the dataset.

- Data sharing policies may change from time to time and new types of privacy attacks to mobility data may be identified, yielding previously released data unprotected. In such events, it is crucial for the data owner to have knowledge of the sensitive information that was leaked, as well as be capable of safeguarding the data based on the new evidence. When the data resides in-house, the privacy-aware query engine can be updated to conform to the new policies and block new types of attack. Additionally, the auditing of queries allows the data owner to have knowledge about the extent of the data leakage by examining the history of user queries to the database.

Data sharing scenario: A data holder, such as a telecom operator or a governmental agency, collects movement information for a community of people. The raw movement data, capturing the location of each individual in the course of time, is processed to generate user trajectories that are subsequently stored in a database. Apart from the analysis that this data undergoes within the premises of the hosting organization, we assume that at least part of the data has to be made available to external, possibly untrusted, parties for querying and analysis purposes. As is evident, direct publishing of this information, even if the data is first deprived from any explicit identifiers, would severely compromise the privacy of the individuals whose movement is recorded in the database. This is due to the fact that malevolent end-users could potentially link the published trajectories to sensitive locations of the individuals (such as their houses), thus identify the users. To ensure privacy-aware sharing of in-house mobility data, a mechanism is necessary to control the information that is made available to external parties when they query the database, so that only nonsensitive information leaves the premises of the hosting organization.

In this paper, we realize and extend the basic design principles that were discussed in [6] by introducing HERMES++, a novel query engine for sensitive trajectory data that allows subscribed end-users to gain restricted access to the database to accomplish various analysis tasks. HERMES++ can shield the trajectory database from potential attacks to user privacy, while supporting popular queries for mobility data analysis, such as range queries, distance queries and nearest neighbors queries. Similarly to [6], HERMES++ operates by retrieving real user trajectories from the database and mixing them with carefully crafted fake trajectories in order to reduce the confidence of attackers regarding the real trajectories in the query result. However, unlike the privacy-engine that was described in [6], HERMES++ achieves to (a) audit end-user queries and effectively block an extended set of attacks to user privacy, securing the database against user identification, sensitive location tracking, and sequential tracking attacks, (b) generate smooth and more realistic fake trajectories that preserve the trend of the original data and use them to augment the real ones in returned answers to queries, and (c) ensure that no sensitive locations that would lead to user identification are reported as part of the returned trajectories. The latter goal is achieved by modifying parts of the trajectories that are close to sensitive locations, such as the houses of the users.
Our work makes the following contributions:

- It improves the design principles of [6] and proposes a full-fledged system that supports privacy-aware sharing of in-house mobility data.
- The proposed fake generation algorithm uses a sophisticated approach to generate smooth, realistic fake trajectories that respect the general trend of the real trajectories in the query result. By preserving the trends of the real trajectories when generating fakes, our framework minimizes the impact of the privacy mechanism to the outcome of the mobility data analysis.
- The proposed auditing mechanism can effectively identify and block a range of potential attacks that could lead to user identification or tracking.

The rest of this paper is organized as follows. Section 2 surveys the related work. In Section 3, we present the types of attacks to user privacy that are blocked by the privacy-aware query engine. Section 4 discusses in detail the auditing and the fake trajectory generation algorithms that are implemented as part of the query engine to support its functionality, while Section 5 sheds light on implementation issues of the engine. In Section 6, we experimentally evaluate the proposed framework demonstrating its effectiveness towards blocking attacks to user privacy, while generating realistic fake trajectories. Section 7 concludes this work.

2. RELATED WORK

Privacy-preserving data publishing has been the focus of attention of the database community for almost three decades. The research in this domain has progressed along two main directions: *providing on-site, restricted access to in-house data* and *providing off-site publication of sanitized data*. In the first category, methodologies have been proposed for disclosure control in statistical databases [2]. These approaches support only count and/or sum queries, since no other information can be made available to the inquirer.

The second direction in privacy-preserving data publishing collects methodologies that provide off-site publication of sanitized data. Several methodologies have been proposed to support different data types and analysis tasks [1,10,11,15–17]. Since our approach aims at concealing sensitive trajectory information, in the following we focus our attention on approaches that protect sensitive mobility data.

Hoh and Gruteser [7] present a data perturbation algorithm that is based on path crossing. The approach identifies when two nonintersecting trajectories that belong to different users are "sufficiently" close to each other in the original dataset and generates a fake crossing of these trajectories in the sanitized counterpart to prevent adversaries from tracking a complete user's trajectory.

Terrovitis and Mamoulis [17] consider datasets that depict user movement in the form of sequences of places that each user has visited, set out in the order of visit. They propose an anonymization approach that suppresses selected places from user trajectories to protect users from adversaries who hold projections of the data on specific sets of places.

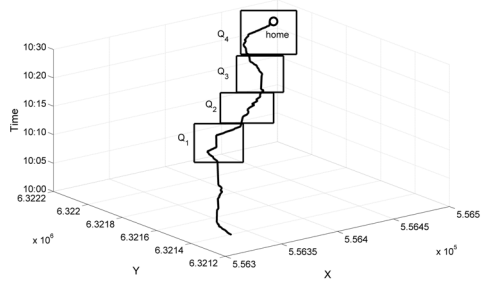

Figure 1: The sensitive location tracking and the sequential tracking attacks to user privacy

Nergiz, et al. [11] also rely on the sequential nature of mobility data and propose a coarsening strategy to generate a sanitized dataset that consists of K-anonymous [15, 16] sequences. The algorithm consolidates the trajectories of the original dataset into clusters of K and then anonymizes the trajectories in each cluster.

Abul, et al. [1] propose a K-anonymity approach that relies on the inherent uncertainty that exists with respect to the whereabouts of the users in historical datasets representing user mobility. The anonymity algorithm identifies trajectories that lie close to each other in time, employs space translation and generates clusters of at least K trajectories. Each cluster of K trajectories forms an anonymity region and the co-clustered trajectories can be released.

The most related work to ours is the study of Gkoulalas-Divanis and Verykios [6], which describes the design principles of a query engine that protects user privacy by generating fake trajectories. The idea behind [6], and also behind this work, is that malevolent users who query the trajectory database should not be able to discover (with high confidence) any real trajectories that are returned as part of the answer set of their queries, while they can use the returned data to support their analytic tasks. A shortcoming of [6] is that the time dimension was not explicitly handled in the overall design. Moreover, the interpolation technique that was proposed for the generation of the fake trajectories was applied on pairs of real trajectories and thus could fail to account for the actual trend in the query region, while also leading to the generation of non-smooth fake trajectories. The current proposal fully implements the proposed privacy engine and, to our knowledge, is the first to introduce a Moving Object Database Engine (MOD) with privacy-preservation functionality.

3. PRIVACY ATTACKS

HERMES++ can effectively protect the privacy of the users by blocking three types of attacks that malevolent users may try to pursue in the original database:

- **User identification attack**: In this attack the identity of the user can be exposed by ad hoc queries involving overlapping spatiotemporal regions.
- **Sensitive location tracking attack**: In this attack the malevolent user tries to map-match one or more locations in a user trajectory to known locations that can effectively expose the identity of the user (e.g., the address of a house or a betting office). We call such locations *sensitive* for the user as they should not be disclosed to the attackers.
- **Sequential tracking attack**: In this attack the user is tracked down through his trajectory by a set of focused queries on regions that are near to each other, in terms of space and time. The attacker can "follow" the user and learn the places that he has visited.

The user identification attack is possible when the query engine answers a query involving a spatial (or spatiotemporal) region and then another, more specific query, involving part of this region. In this case, the attacker can breach the enforced privacy model by identifying fake trajectories which, in turn, increases his confidence regarding the real trajectories in the system. To block this attack, we use auditing to track the queries initiated by each end-user in the system and deny answering overlapping queries.

The sensitive location tracking attack allows malevolent users to distinguish real trajectories from fakes, learn sensitive locations that real users have visited, and (possibly) reveal the identity of these users. To block these attacks, we protect the starting and the ending location of trajectories, as well as any other (owner-specified) location in the course of the user trajectory that can be considered as *sensitive* for the user. As an example of this type of attack, assume a query that involves region Q_4, shown in Figure 1. Since in this region the trajectory has its end point to a sensitive location, the attacker can map-match this location and reveal the user's identity. The attack can succeed even if fake trajectories are generated in this region because the probability of a fake trajectory having an end point to a sensitive location is low, while this is very common for real users. To block the sensitive location tracking attack, our auditing approach identifies sensitive locations of trajectories that appear in the query window and proceeds to dislocate them so that the sensitive location is not disclosed.

Last, in the sequential tracking attack an attacker attempts to "follow" a user trajectory in the system by using a set of focused queries involving spatiotemporal regions that are adjacent to each other. To block this attack, the proposed auditing algorithm takes the necessary measures to smoothly continue the movement of fake trajectories from neighboring regions (returned as part of previous queries of the user) to the current region, so as to prohibit attackers from distinguishing the fake trajectories from the real ones.

4. ALGORITHMS

In the following sections we present the algorithms that deliver the functionality of HERMES++. Section 4.1 describes the algorithm that we designed for the generation of realistic fake trajectories. In Section 4.2, we present the auditing technique that is used to audit user queries and preserve the privacy in the answers to the queries.

4.1 Fake trajectory generation

The proposed fake trajectory generation algorithm has the ability to produce trajectories that follow the trend of the input set of real trajectories, thus minimize the potential of privacy breaches when query results are released to the end-users. This algorithm plays a central role in our privacy-aware query engine. When a user poses a query to the database, the engine provides the answer only if at least

Algorithm 1 Fake Trajectory Generation

```
1: function FAKE-GEN(line segments S_i, minimum number of points
   MinLns, smoothing parameter γ, time step of sampling rate
   Timestep, MBB(t_MBBmin,t_MBBmax), d_min, d_max, l_min, l_max,
   l_avg, avgU_min, avgU_max)
2:     fake_trajectory ← RTG(S_i, MinLns, γ)
3:     calculate initial timestamp t_0 of the fake trajectory
4:     t_max ← t_0 + |fake_trajectory| * Timestep
5:     if (t_max > t_MBBmax) then
6:         repeat
7:             Douglas_Peucker (fake_trajectory, f)
8:             f ← f/2
9:             t_max ← t_0 + |fake_trajectory| * Timestep
10:        until (t_max < t_MBBmax)
11:        if (l_max > 2 * l_avg) then
12:            l_max ← random(l_avg, 2 * l_avg)
13:        else
14:            l_max ← l_avg* random(1, l_max/l_avg)
15:    for each (p_i ∈ fake_trajectory) do
16:        set timestamps of initial and final point of p_i
17:        calculate speed U_i of p_i
18:        if (U_i < avgU_min or U_i > avgU_max) then
19:            repeat
20:                l ← random(l_min, l_max)
21:                calculate new speed U_i of p_i
22:            until (U_min < U_i < U_max)
23:        calculate angle φi
24:        define coords of new ending point based on l
25:        map match p_i
26:    return fake_trajectory
```

L real user trajectories exist in the area. The lower bound L in the number of users is a simple way to prevent answering queries whose original result set is very small (e.g., a range query in a region with very few trajectories), as in this case the generated fake trajectories may fail to capture the trend of the real trajectories. Prior to releasing any real trajectory, an approach is employed (*see* Section 4.2) to protect any sensitive locations in the trajectory that could be used by malevolent end-users to identify the corresponding user. To produce the answer set for the query, the engine generates N fake trajectories, where N is an owner-specified threshold. The proposed algorithm has the ability to produce fake trajectories for different types of queries, such as range, nearest neighbor and distance queries, while it is used by our auditing mechanism (Section 4.2) to handle different types of attacks from malevolent users.

Unlike the simple fake generation approach of [6], the fake trajectory generation algorithm that we propose in this work is based on the idea of the *Representative Trajectory Generation* (*RTG* for short) algorithm, introduced by Lee et al. in [9]. The main idea of this algorithm is that the resulting representative trajectory describes the overall movement of a set of directed segments, produced after the partitioning of a set of trajectories. The partitioned trajectories (i.e., directed segments) are clustered according to a distance function taking into account the parallel, perpendicular and angle distance of the segments. The outcome of the *RTG* algorithm, applied on each cluster, produces a smooth (more or less) linear trajectory that best describes the corresponding cluster. However, similarly to [6], the *RTG* algorithm also fails to consider the temporal dimension of the generated trajectory. As a result, our proposed fake trajectory generation algorithm has to transform the *RTG* output by appropriately integrating the time dimension into the fake trajectory generation process.

Algorithm 1 provides the details of our fake trajectory generation approach. The algorithm takes as input a set of

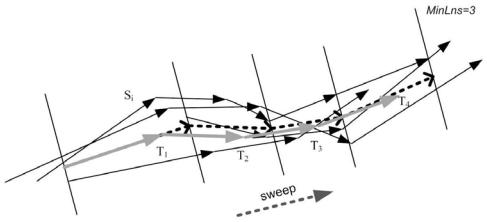

Figure 2: Generating a fake trajectory based on the representative trajectory of a set of line segments

line segments S_i resulting from a set of trajectories which form the answer to a user query. In the first step (line 2), the representative trajectory is produced based on this set of line segments of trajectories. For simplicity reasons, in Figure 2 we depict segments as consecutive parts of trajectories; however, in the general case, they could be disconnected and independent segments that are filtered in a way that all move towards (more or less) the same direction. This is because the *RTG* algorithm assumes that all segments follow the same directional pattern. In the sequel, the *RTG* algorithm sweeps a rotated vertical line according to the average direction vector towards the major axis, counting the number of line segments that are either the starting or the ending point of a line segment. If the resulted number is equal to or greater than a threshold *MinLns*, the algorithm calculates the average coordinate of those points and assigns the average into the set of representative trajectory; otherwise, it proceeds to the next point. To avoid segments that are too close to each other, a smoothing parameter γ is utilized. The final outcome of this step is the trajectory with the dotted line shown in Figure 2.

After calculating the representative trajectory (line 2), the algorithm inserts the time dimension to each line segment and performs additional computations to adjust it and make it more plausible. In detail, we examine and require for a realistic length and speed for the 3D segments of the fake trajectory. If these measures get unusual values then an adversary may be able to identify which trajectory is the fake one. In Figure 2, the grey solid line depicts the final fake trajectory after assigning the time dimension to the segments and adjusting them to be more realistic. In order to achieve this, the algorithm takes as parameter the spatiotemporal *Minimum Bounding Box* (MBB), which is set by the auditing mechanism and may be either the MBB of the user's query parameter (in the case of range queries), or the MBB that is formed by the whole trajectories whose parts belong to the results of user's query. An additional set of input parameters that is provided by the auditing mechanism corresponds to statistical computations regarding $d_{min}, d_{max}, l_{min}, l_{max}$, which are the minimum and maximum trajectories' duration and segments' length, respectively, and $avgU_{min}, avgU_{max}, l_{avg}$, which are the average minimum and maximum speed, as well as, the average length of the segments, respectively. The *Timestep* parameter is the duration of a line segment and is considered to be constant indicating that the moving object transmits its location update at regular temporal intervals. The outcome of the algorithm is a set of line segments forming a trajectory, which are stored in the array *fake_trajectory*.

Having calculated the set of line segments, the algorithm computes the initial timestamp t_0 that the fake trajectory will start at (line 3). The initial timestamp is defined as $t_0 = t_{MBBmin} +$ random(0,SP), where $SP = (t_{max} - t_{min}) - random(d_{min}, d_{max})$ corresponds to a value used to ensure that time t_0 of the first point of the fake trajectory will not be placed near t_{MBBmax}. Moreover, the maximum timestamp of the fake trajectory should not exceed t_{MBBmax}, otherwise it will differ from the real trajectories. In order to ensure this, the maximum timestamp t_{max} of the fake trajectory is calculated (line 4) as a function of the initial timestamp t_0 and the duration of the fake trajectory (i.e., $|fake\_trajectory| * Timestep$). If ($t_{max} > t_{MBBmax}$) then a line simplification procedure is applied to reduce the number of line segments (lines 6-10). *Douglas-Peucker* (line 6) [5] is an algorithm that compresses the generated segments by using a polyline representation and a parameter f that corresponds to a distance threshold, defined as a percentage of the trajectory's length. The compression procedure is repeated until ($t_{max} < t_{MBBmax}$) and in each iteration the parameter f is halved.

Having calculated the initial timestamp, the algorithm adjusts the maximum length l_{max} of the segments that have been generated (lines 11-14) in order to manipulate long segments that will lead to the generation of non-realistic fake trajectories. Specifically, if l_{max} is greater than twice the average length l_{avg}, then l_{max} is being recalculated as a random value between l_{avg} and the twice of l_{avg}. Otherwise, the algorithm sets l_{max} randomly between l_{avg} and l_{max}. Then, the algorithm enters a loop (line 15) and assigns the time dimension to each line segment of the fake trajectory. The initial timestamp t_0 of the first line segment has been calculated in previous steps. The timestamp of the ending point of this segment equals to t_0 increased by the sampling rate's duration, i.e., is equal to $t_0 + Timestep$. The ending timestamp of the initial segment will be the starting timestamp of the next segment. Generally, for each line segment it holds that $t_{i+1} = t_i + Timestep$, where $0 \leq i < |fake\_trajectory|$.

After assigning the time dimension to the current segment p_i (line 16), the algorithm proceeds to calculate the speed U_i for each segment p_i (line 17) and checks if it lies within $avgU_{min}$ and $avgU_{max}$ (lines 18-22). If it is outside this range, the algorithm calculates a random segment length l, between l_{min} and l_{max}, such that the speed U_i of the specific segment is within the limits. As a final step, the coordinates of the new ending point are identified based on the length of segment l that was calculated before (lines 23-24).

Depending on the direction of the segment and its angle ϕ_i with the x-axis, the fake trajectory generation algorithm calculates the new coordinates (x_{t+1}, y_{t+1}). The angle ϕ_i is given by $\phi_i = \text{atan2}(y_{t+1} - y_t, x_{t+1} - x_t)$, while the new coordinates (x_{t+1}, y_{t+1}) are calculated as:$x_{t+1} = x_t + l * cos(\phi)$ and $y_{t+1} = y_t + l * sin(\phi)$, where l is the length of the line segment. In the case that trajectory data are related to an underlying road network, the fake trajectory generation algorithm map matches the generated fake trajectory with the specific road network by employing a map matching algorithm (line 25) [3]. This functionality of the algorithm can lead to a more realistic representation of the fake trajectory and thus disincline adversaries from identifying real users. After calculating the new coordinates the algorithm proceeds to the next segment and the procedure continues until all line segments are examined. Finally, the generated fake trajectory is returned (line 26). Figure 3 demonstrates the outcome of the *Fake-Gen* algorithm (green trajectory) with respect to the *RTG* algorithm (red trajectory), when applied to a set of trajectories (i.e. blue trajectories). Obviously, the result of *Fake-Gen* produces a more realistic representation of the trend of the real trajectories.

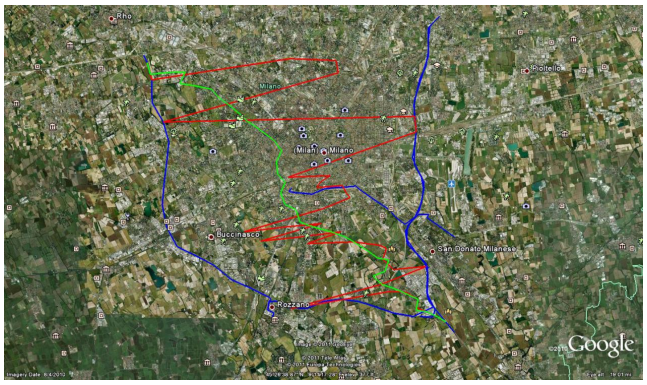

Figure 3: The outcome of *Fake-Gen* (green trajectory) w.r.t. the *RTG* algorithm (red trajectory) when applied on a set of trajectories (shown in blue)

4.2 Query auditing

Algorithm 2 presents our query auditing approach for shielding the database against malevolent queries. When a new query is submitted to the engine, the auditing algorithm first examines if this query involves an area that (partially) overlaps with that of a previous query, submitted by the same end-user. If this is the case, then it denies servicing the query (lines 2-3) to block a potential user identification attack. If the previous test is negative, the auditing mechanism executes the actual query of the user and retrieves the result set (line 4). In order to prohibit the identification of an individual by an adversary that is able to link sensitive locations that are visited by a user (e.g., the home of the user) with trajectories that belong to the specific query, we propose the *Hide Sensitive Location Algorithm* (line 5). This algorithm takes as input a set of sensitive locations SL, a set of trajectories T and the MBB formed by user's query. Initially, the algorithm selects all sensitive locations SL' that lie inside the MBB (line 2 of Algorithm 3). For each trajectory of the given set T, it defines those sensitive locations, SL'_i, that correspond to the current trajectory (lines 3-4). For every sensitive location, $SL'_{i,j}$, it examines if fake sub-trajectories that hide the sensitive locations have been previously computed for this trajectory and retrieves them from *History* (lines 5-7). Otherwise, it computes a new synthetic (fake) trajectory that is then stored for future reference (lines 9-14).

The procedure that is followed by Algorithm 2 to generate and update the synthetic sub-trajectory is based on a variant of the GSTD trajectory synthesizer, called GSTD*, proposed in [13]. GSTD* produces trajectories following complex mobility patterns based on a given distribution of spatiotemporal focal points, to be visited by each trajectory in a specific order. The general idea behind the generator is to use the focal points so as to attract each trajectory's movement. When a particular trajectory has reached the area around a focal point, having at the same time com-

pleted the respective temporal predicate, the generation algorithm changes the attracting point to the next focal point in the list, and so on, until no focal points are left unvisited.

The idea of hiding sensitive locations of a trajectory by misplacing its route is illustrated in Figure 4. The algorithm discovers the intersection points of the trajectory with a circle that is formed around a sensitive location by taking as radius the distance between the sensitive location from a point where the object would have been moved after a certain period of time tw (i.e., tw is a temporal window), if it was moving with its current speed. The idea is to use these intersection points as the focal points in GSTD* (line 9) (see the filled gray circles in Figure 4). If the number of focal points is greater than two (i.e. the object enters and/or leaves the circle more than two times), the algorithm utilizes the first (entering) and the last (leaving) one. In case where the sensitive location is either the initial or the ending point causing the creation of only one focal point, the algorithm randomly selects another random focal point in the perimeter of the circle (line 10-11). After determining focal points it produces a synthetic (fake) trajectory by applying GSTD* between the two chosen focal points as illustrated in the figure with the dotted line (line 12). The algorithm returns the set of trajectories that does no longer contains sensitive locations (line 15).

Having protected the sensitive locations of the trajectories in the querying region, Algorithm 2 commands the generation of the necessary fake trajectories for this region (lines 12-22). To generate the requested number of fake trajectories, the algorithm calculates a set of basic statistics (line 12) that are needed by the fake trajectory generation approach (Algorithm 1), while trying to find trajectories that follow more or less the same direction in the query region (lines 13-21). Specifically, a step dir_{step} (in degrees) is randomly selected (line 14) in the rage of $(0, dir_{step_{max}})$, with $dir_{step_{max}}$ being an input parameter that defines the size of an angular range used to divide the Cartesian plane. As illustrated in Figure 5, the algorithm selects those segments from the real trajectories that belong to the range (dir_{min}, dir_{max}) (see the solid lines in the figure), which are set by randomly assigning dir_{min} and then setting dir_{max} equal to $dir_{min} + dir_{step}$. Subsequently, it calls Algorithm 1 on these segments and passes the query window to create one new fake trajectory. The same process is repeated for the next range of directions, which leads to the generation of another fake trajectory, until the 360° are exceeded. Then, the algorithm selects a new dir_{step} and repeats the same process, until the requested number of fake trajectories are generated (line 22). Note that the filtering approach on the directional property of the segments guarantees that the fake generation algorithm will produce nice representative trajectories of the query result, as it acts as a simple clustering methodology on the overall set of available segments.

After generating the fake trajectories, Algorithm 2 takes the necessary measures to protect the privacy of the users whose movement is depicted in the query window by smoothly continuing the movement of the fake trajectories from neighboring regions, returned as part of previous queries posed by the end-user, to the current one. Specifically, the algorithm examines if the query posed by the end-user has a nearby query made by the same end-user in the past, which does not exceed a spatial s_{thr} and a temporal t_{thr} threshold. In case that the query has only one such neighbor, the algorithm performs a one-by-one matching (line 24) between the fake trajectories of MBB and MBB_{hist} (i.e. the nearby query saved in *History*). In detail, it first finds the MBB with the minimum number of fake trajectories and then it randomly matches each one of them with fakes from the other query, by producing pairs P_i of fake trajectories. For each pair, it examines if MBB touches MBB_{hist} or if they are apart. In the first case, illustrated in Figure 6, a space time translation is performed to connect the two fake trajectories. The fake trajectory is transferred in the x and y axes, if necessary. Then, the algorithm checks the time dimension to assure there is no temporal gap. If such a gap exists, the algorithm recalculates the timestamp of each point of the fake trajectory. In the second case (see Figure 7), where a spatial and/or a temporal gap exists between MBB and MBB_{hist}, Algorithm 2 generates a connection-trajectory (see the dotted lines) between them by using the GSTD* algorithm [13]. Focal points are the ending point of the one trajectory with the starting point of its matching trajectory in P_i. After generating the fake trajectories, the algorithm applies the hiding process of the sensitive locations also for these trajectories (line 32), to conceal the fact that they are fakes.

The trajectory auditing algorithm (i.e., *TrajAuditor*), as presented in Algorithm 2, covers the cases of range and distance queries. The same algorithm can be minimally adapted to support k-nearest neighbors queries. In the case of k-nearest neighbors queries, the end-user has to provide as parameters the value of k, the trajectory for which he or she searches its k nearest neighbors, and a temporal period which is a sub-period of the trajectory's lifespan. Moreover, the MBB window in a k nearest neighbors query refers to the MBB of the trajectories that form the result of this query. Thus, to support such queries, Algorithm 2 has to be modified so that in lines 12 (17), where statistics (MBB parameter for Algorithm 1) for the generation of the fake trajectories are computed, the window will be the MBB of all real trajectories in the query result.

As a last remark, we need to point out that Algorithm 2 commands the generation of the necessary number of fake trajectories based on the parts of the real trajectories that appear inside the query window. An alternative approach (henceforth called *TrajFaker*) would be to generate wide fake trajectories that exceed the limits of the window the user submitted. In this case, auditing would still be applicable but not forced, contrary to the case of Algorithm 2. The alternative strategy differs from that of Algorithm 2 in the following steps. When a user executes a query, the new approach finds the trajectories that are contained in the specific spatiotemporal window (or the k nearest neighbors in case of such queries) and then retrieves the whole trajectories and not the parts of them that lie inside the window. Subsequently, it generates fake trajectories by employing Algorithm 1 on the whole trajectories. Each generated fake trajectory is examined to see whether it crosses the spatiotemporal window of the query and, if so, it is included to the returning set. Otherwise, the trajectory is discarded and the same process is repeated. All generated fake trajectories are stored in order to participate to the generation of other fake trajectories. Finally, there are no privacy threats with respect to sequential tracking as before, since the generated fake trajectories are based on the whole trajectories and not parts of them, and are stored. If an adversary tries to exe-

Algorithm 2 Query Auditing Algorithm

function TRAJAUDITOR(user's query MBB, number of generated fake trajectories N, lower bound threshold L, spatial threshold s_{thr}, temporal threshold t_{thr}, maximum direction step $dir_{step_{max}}$, set of sensitive locations SL, temporal window tw, $Minlns$, γ, $Timestep$)
  if (CHECKHISTORY(User has posed an overlapping query w.r.t. MBB) = true) **then**
    Privacy threat: Overlapping queries
  $TR \leftarrow$ SPATIOTEMPORALRANGEQUERY(MBB)
  $TR \leftarrow$ HIDESENSITIVELOCATIONS(SL, TR, MBB, tw)
  if (CHECKHISTORY(User has posed in the past a nearby query w.r.t. s_{thr}, t_{thr}) = true) **then**
    Privacy threat: Sequential tracking attack
  else
    if ($|TR| \leq L$) **then**
      Privacy threat: Lower bound threshold violation
    else
      CALCULATESTATISTICS($d_{min}, d_{max}, l_{min}, l_{max}, l_{avg}, avgU_{min}, avgU_{max}$)
      repeat
        $dir_{step} \leftarrow$ random$(0, dir_{step_{max}})$
        $dir_{min} \leftarrow$ random$(0, 360)$; $dir_{max} = dir_{min} + dir_{step}$
        repeat
          $S_i \leftarrow$ FILTER_BY_DIRECTION(dir_{min}, dir_{max}, TR)
          $FT \leftarrow FT \cup$ FAKE_GEN($S_i, MinLns, \gamma$, $Timestep$, MBB, $Statistics$)
          $dir_{min} \leftarrow dir_{min} + dir_{step}$
          $dir_{max} \leftarrow dir_{max} + dir_{step}$
        until ($dir_{max} > 360$)
      until ($|FT| = N$)
  Retrieve from $History$ all fakes FT_{hist} from a nearby query of the user w.r.t. s_{thr}, t_{thr}
  $P_{match} \leftarrow$ MINRANDOMMAX(FT, FT_{hist})
  for each pair ($P_i(T_j, T_k) \in P_{match}$) **do**
    if (MBB touches a historic query of the user) **then**
      SPACE_TIME_TRANSLATION(P_i)
    else
      $focal_{points} \leftarrow (T_{j_{end}}, T_{k_{start}})$
      GSTD*($focal_{points}$)
      Update in FT the fake trajectory that corresponds to P_i
  $FT \leftarrow$ HIDESENSITIVELOCATIONS(SL, FT, MBB, tw)
  UPDATEHISTORY
  return ($TR \cup FT$)

Algorithm 3 Hide Sensitive Locations Algorithm

function HIDESENSITIVELOCATIONS(set of sensitive locations SL, set of trajectories T, user's query MBB, temporal window tw)
  $SL' \leftarrow SL$ inside MBB
  for each ($T_i \in T$) **do**
    $SL'_i \leftarrow$ select the subset of SL' that corresponds to T_i
    for each (sensitive location of T_i, $SL'_{i,j} \in SL'_i$) **do**
      if (a fake sub-trajectory has been computed in the past for this $SL'_{i,j}$) **then**
        Retrieve the fake sub-trajectory from $History$ and update T_i
      else
        $focal_{points} \leftarrow$ INTERSECTION(T_i, buffer($SL'_{i,j}, tw$))
        if ($|focal_{points}| = 1$) **then**
          $focal_{points} \leftarrow$ ADDRANDOMPOINTONSURFACE(buffer(T_i, tw))
          Produce a synthetic/fake trajectory by applying $GSTD^*$ between first and last $focal_{points}$
          Update the part of T_i with the synthetic/fake sub-trajectory
          UPDATEHISTORY
  return (T)

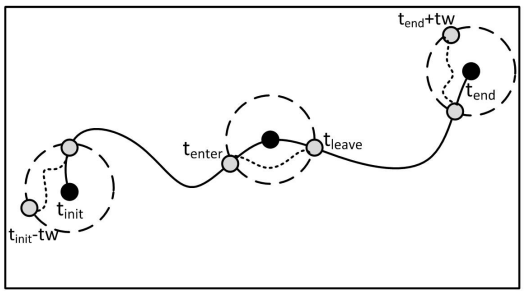

Figure 4: Protecting the sensitive locations of user trajectories

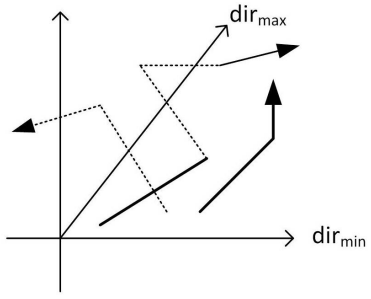

Figure 5: Selecting segments from real trajectories

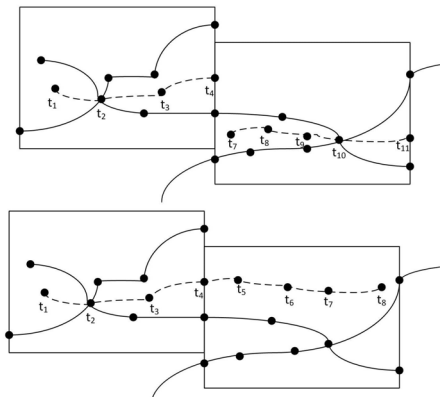

Figure 6: Prohibiting sequential tracking (Case I)

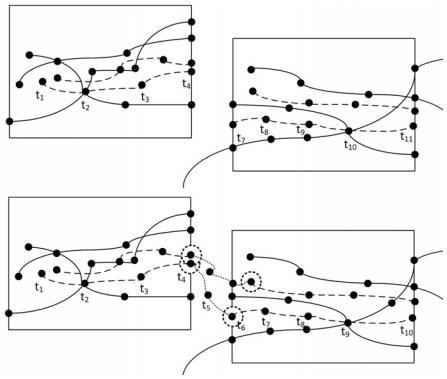

Figure 7: Prohibiting sequential tracking (Case II)

cute overlapping or sequential queries, the fakes will appear in all of these queries' answers.

5. SYSTEM IMPLEMENTATION

HERMES++ has been designed as a system extension of the HERMES query engine [12], which in turn was developed as an extension that provides trajectory functionality to Oracle's Object-Relational DBMS (ORDBMS). HERMES++ exploits on the trajectory storage functionality and the spatiotemporal query processing capabilities of HERMES for providing anonymous queries to users. More specifically, HERMES defines a trajectory data type and a collection of operations as an Oracle data cartridge, which is further enhanced by a special trajectory preserving access method, namely the TB-tree [14]. HERMES++ directly utilizes this functionality at the ORDBMS level to store real and fake trajectories, as well as any historic information of all the users' queries (and the corresponding responses), in order to avoid different types of tracking attacks (e.g., sequential tracking). It succeeds so by an embedded auditing module which invokes the HERMES's queries and the fake trajectory generator algorithm. The whole framework is build at the ORDBMS level, which means that HERMES++'s users have the ability to pose their queries through a PL/SQL interface. As such, from an architectural point of view, HERMES++ acts as a wrapper over the HERMES query engine and not as a secure middleware. Figure 8 illustrates HERMES++ architectural framework.

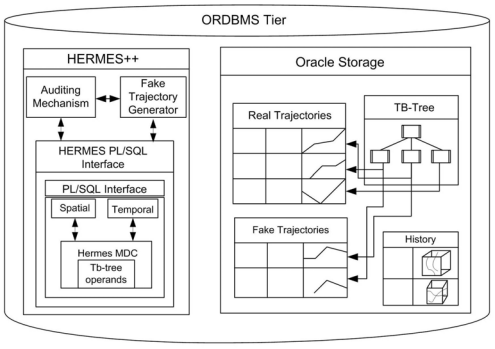

Figure 8: HERMES++ system architecture

6. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the proposed privacy-aware query engine to assess its effectiveness in protecting user privacy when answering queries. To evaluate the distortion that is caused by the query engine to the database due to the generation of fake trajectories, we consider that the data owner *requires that at least K trajectories are returned to the end-users in response to their queries*, for all different types of supported queries. Parameter K is similar to using a variable N threshold for the generation of fake trajectories; it's purpose is to provide insight on the performance of the query engine when the number of generated fake trajectories is dynamically changing. To assess the efficiency of the proposed auditing methodology, we compute the time lag in the responses to the end-users' queries. We report experiments over a synthetic dataset, which was generated using Brinkhoff's network-based generator for moving objects [4]. The dataset contains 4495 trajectories, all living in a temporal period of 800 timestamps. A location update is performed in successive timestamps that are 5 minutes far from each other. The dataset has been generated over the road network of Oldenburg (Germany). It contains 153163 points, the radius area containing all trajectories is 17889.6, the maximum length of a trajectory is 668 and the average distance between consecutive points is 697.9.

For each query type that is supported by our query engine, we evaluated our methodology by measuring the database distortion that is defined as the percentage of the generated fake trajectories with respect to the size of the database. For each experiment, we created 1000 random queries. For range queries (and distance queries, which are handled as range queries with the distance threshold being the measure for determining the range size), we randomly selected different sizes, locations and time durations, while for k nearest neighbors queries we randomly selected the query trajectory and the temporal period (sub-period of the query trajectory's lifespan) for different values of k. For k nearest neighbors queries the corresponding MBB was set to $0.1 * d$, where d is the length of the maximum side of the MBB of the whole dataset. This implies that a trajectory is not included in the k nearest neighbors if its distance is greater than $0.1 * d$. We use this approach to exclude results sets that cover a disproportionally large MBB that will have as an effect the creation of many fakes. In all cases, we have set the $MinLns$ parameter to 2, the γ smoothing parameter to 5m, and the *Timestep* to the sampling rate of the dataset.

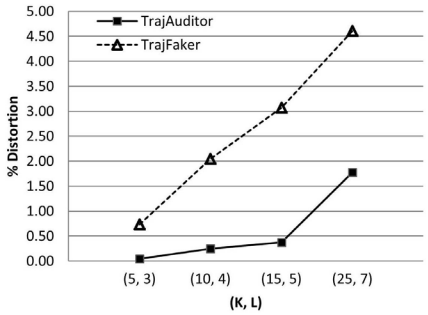

Figure 9: Distortion by range queries w.r.t. K, L

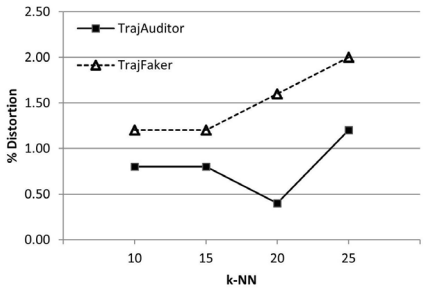

Figure 10: Distortion by kNN queries w.r.t k ($K = 5$)

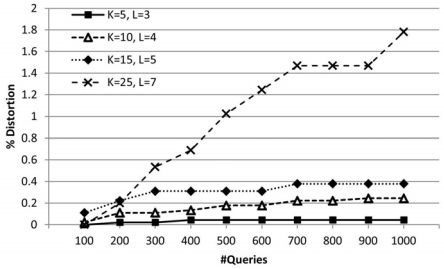

Figure 11: Distortion over time w.r.t. the value of K - *TrajAuditor*

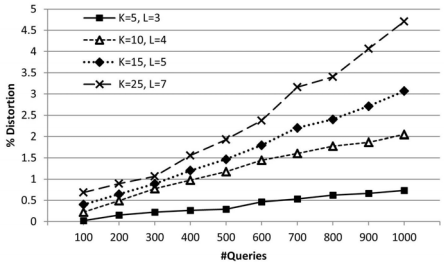

Figure 12: Distortion over time w.r.t. the value of K - *TrajFaker*

Moreover, we have set the spatial (temporal) threshold s_{thr} (t_{thr}) to $0.001 * d$, respectively, while the maximum direction step $dir_{step_{max}}$ was set to 45°.

Figure 9 illustrates the distortion of the database for spatiotemporal range queries for both proposed approaches. Specifically, the figure shows the percentage of the generated fake trajectories for different value-pairs of K and L. Note that we slightly increase the L bound analogously with the increase of the K threshold. Clearly, the overall distortion of the database even for large thresholds of K (w.r.t the size of the database) is very low. Moreover, the *TrajAuditor* results in less distortion for small values of K, while for larger values the relative rate of distortion w.r.t. *TrajFaker* is decreased. This is rational, as for larger K thresholds the *TrajFaker* will produce less fakes, but longer, which have been generated in preceding queries. Intuitively, the fakes are less in this case as the algorithm re-uses parts of the long fakes produced by other queries.

In Figure 10 we repeat the same experiment for kNN queries but this time we keep the K threshold stable to 5 and scale the number of k nearest neighbors. As expected, again the results present the same pattern. In the sequel and due to space limitations, we omit further presentation regarding k nearest neighbors queries as they do not allow different conclusions from those of range queries.

Figures 11 and 12 show the distortion in the database for *TrajAuditor* and *TrajFaker* (respectively) at any given time. The distortion is plotted with respect to the users' queries (in temporal order, but still in random sizes and locations), for different values of K. As expected, lower values of K result in a smaller distortion of the database, since less fakes are generated. Furthermore, for any value of K the curve is increasing, however, for the case of *TrajAuditor* it is convex, which means that there is some time instance at which the fakes in the dataset suffice to answer a reasonable amount of user queries, without the need of generating additional fakes. On the other hand, in the case of *TrajFaker*, the curve is monotonously increasing, but note that in this case we have not applied any of the auditing mechanisms, so we respond to any user query by generating fakes. Despite this the distortion remains at low levels, lower than 5%, even for $K = 25$. The reason that the database distortion is low, relies on the re-use of the fake trajectories across different users' queries. An additional qualitative advantage of the *TrajFaker* is that fake trajectories in this case are more realistic, as they are produced by whole, real trajectories and not portions whose size depend on the query size.

Next, we conducted experiments over range queries of varying volumes (i.e. multiples of the whole space-time). The queries are clustered in ten equally sized groups. Each group contains randomly selected queries of the same volume. Figure 13 depicts the distortion in the database when applying randomly 1000 queries using *TrajFaker*. Notice the decrease in database distortion as the size of the queries increases. This is because large queries will not be forced to generate more fakes, since some of the fakes will have already been created by small queries that produce long fakes.

Figure 14 presents the number of query rejections (denials) of the *TrajAuditor* for different values of K, and L. The upward convex trend of the curves is due to the fact that as the volume of the queries increases it is more probable to lead to sequential tracking attacks. We note that larger K values result in less query rejections. This is justified by the simultaneous increase of the L bound. A larger lower bound L results in more query denials. More such

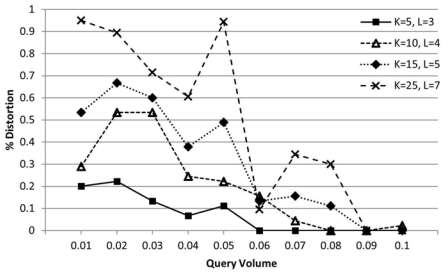

Figure 13: Distortion vs. query size w.r.t. the value of K - *TrajFaker*

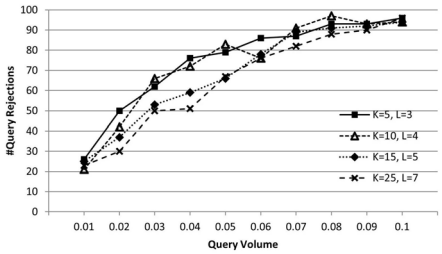

Figure 14: Query rejections vs. query size w.r.t. K

rejections mean fewer MBBs to be stored in history, which means fewer rejections due to sequential tracking attacks.

Last, we study the efficiency of the auditing mechanism. Figure 15 presents the average time over all 1000 queries of how much does *TrajAuditor* (*TrajFaker*) delay the response to the user's query. As expected, *TrajAuditor* presents larger time lags, while both approaches show a superlinear behavior due to the generation of fakes.

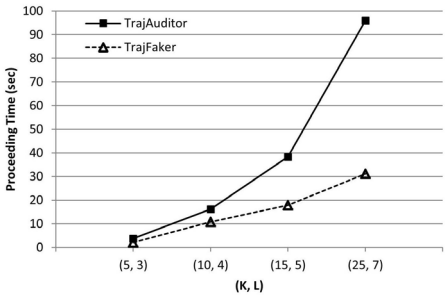

Figure 15: Time lag w.r.t the value of K

7. CONCLUSION

In this paper we presented HERMES++, a privacy-aware query engine that enables the remote analysis of user mobility data. HERMES++ supports a variety of popular spatial and spatiotemporal queries and uses auditing and fake trajectory generation techniques to identify and block, respectively, potential attacks to user privacy. Through experimental evaluation, we demonstrated the effectiveness of our approach to protect the privacy of the users, while minimally distorting the mobility dataset.

Acknowledgments

Research partially supported by the FP7 ICT/FET Project MODAP (Mobility, Data Mining, and Privacy) funded by the European Union. URL: `www.modap.org`.

8. REFERENCES

[1] O. Abul, , F. Bonchi, and M. Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *ICDE*, pages 376–385, 2008.

[2] N. R. Adam and J. C. Worthmann. Security–control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map–matching vehicle tracking data. In *VLDB*, pages 853–864, 2005.

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

[5] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, 1973.

[6] A. Gkoulalas-Divanis and V. S. Verykios. A privacy–aware trajectory tracking query engine. *SIGKDD Explorations*, 10(1):40–49, 2008.

[7] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. In *SECURECOMM*, pages 194–205, 2005.

[8] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Preserving privacy in GPS traces via uncertainty–aware path cloaking. In *CCS*, pages 161–171, 2007.

[9] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.

[10] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE*, page 25, 2006.

[11] M. E. Nergiz, M. Atzori, and Y. Saygin. Towards trajectory anonymization: A generalization–based approach. In *ACM GIS Workshop on Security and Privacy in GIS and LBS*, pages 1–10, 2008.

[12] N. Pelekis, E. Frentzos, N. Giatrakos, and Y. Theodoridis. HERMES: Aggregative LBS via a trajectory DB engine. In *SIGMOD*, pages 1255–1258, 2008.

[13] N. Pelekis, I. Kopanakis, E. E. Kotsifakos, E. Frentzos, and Y. Theodoridis. Clustering uncertain trajectories. *KAIS*. to appear.

[14] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, 2000.

[15] P. Samarati. Protecting respondents' identities in microdata release. *TKDE*, 13(6):1010–1027, 2001.

[16] L. Sweeney. $\mathcal{K}$–anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge Based Systems*, 10(5):557–570, 2002.

[17] M. Terrovitis and N. Mamoulis. Privacy preservation in the publication of trajectories. In *MDM*, pages 65–72, 2008.