Chapter 11
Systems for Privacy-Preserving Mobility Data Management

Despina Kopanaki, Nikos Pelekis, and Yannis Theodoridis

Abstract The increasing availability of data due to the explosion of mobile devices and positioning technologies has led to the development of efficient management and mining techniques for mobility data. However, the analysis of such data may result in significant risks regarding individuals' privacy. A typical approach for privacy-aware mobility data sharing aims 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 violation. On the other hand, an alternative strategy considers that data stays in-house to the hosting organization and privacy-preserving mobility data management systems are in charge of privacy-aware sharing of the mobility data. In this chapter, we present the state-of-the-art of the latter approach, including systems such as HipStream, Hermes++, and Private-Hermes.

11.1 Introduction

Recent 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 the 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

D. Kopanaki (✉) · Y. Theodoridis
Department of Informatics, University of Piraeus, Piraeus, Greece
e-mail: dkopanak@unipi.gr; ytheod@unipi.gr

N. Pelekis
Department of Statistics & Insurance Science, University of Piraeus, Piraeus, Greece
e-mail: npelekis@unipi.gr

A. Gkoulalas-Divanis, C. Bettini (eds.), *Handbook of Mobile Data Privacy*,
https://doi.org/10.1007/978-3-319-98161-1_11

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.

In this chapter, we consider the following data sharing scenario: a data holder (telecom operator, governmental agency, etc.) collects movement information about 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.

Recently, several methodologies have been proposed to enable privacy-preserving mobility data sharing. Existing approaches, such as [1, 2, 9, 10, 16, 30], 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 stored in the original dataset can be disclosed without causing any privacy violations. However, this assumption can be proven unrealistic in certain data sharing scenarios. In order to avoid privacy breaches a more conservative approach can be employed by assuming that the majority of the information that is captured in the mobility dataset should remain private and that the data should stay in-house to the hosting organization. This 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 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 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 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 and keeping track of the returned answers.

In this chapter, we present the state-of-the-art systems which are based on the assumption that data should stay in-house to the hosting organization in order to ensure that no privacy violation may occur during analysis processes. First, Gkoulalas-Divanis and Verykios [8] proposed a query engine that offers k-anonymous answers to user queries. The engine generates fake records to guarantee about what can be found by untrusted third parties. Based on the same notion, Hermes++ which was proposed by Pelekis et al. [22], is 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 neighbor queries. Hermes++ operates by retrieving real user trajectories from the database and generating carefully crafted fake trajectories in order to reduce the confidence of attackers regarding the information of the real trajectories in the query result. Hermes++ achieves to (a) audit end-user queries and 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 (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.

Moreover, we present Private-Hermes [23], a benchmark framework for privacy-preserving mobility data querying and mining methods. The first dimension of this benchmark with respect to privacy issues involves in-house stored data and privacy-aware query answering. Private-Hermes incorporates Hermes [21], a query engine based on a powerful query language for trajectory databases, which enables the support of aggregative queries. Hermes supports a variety of well-known queries such as range, nearest neighbor, topological, directional queries, etc. On top of this functionality, Hermes++ audits queries for trajectory data to block potential attacks to user privacy, supports the most popular spatiotemporal queries (range, distance, $k - NN$) and preserves user privacy by generating carefully crafted, realistic fake trajectories. The second dimension with respect to privacy that is supported by this benchmark involves privacy-preserving MOD publishing. Two state-of-the-art algorithms, namely NWA [1] and W4M [2], have been integrated in Private-Hermes to help anonymize trajectories. The objective is to support the evaluation of such

anonymization techniques and to study their effect in the utility of the sanitized data, when compared with queries into the original MOD.

Finally, HipStream [31] is a privacy-preserving system for managing mobility data streams. The system enforces three fundamental Hippocratic principles introduced by Agrawal et al. [4] of limited use, limited disclosure and limited collection of data during data stream management. Hippocratic databases extend the functionalities of traditional databases with privacy-preserving capabilities. Service providers have limited access to the data w.r.t. the privacy requirements that the data owner has enforced. Queries are modified if needed from the system and data are partially anonymized if necessary before being processed.

The rest of this chapter is organized as follows. Section 11.2 provides a description of the background of privacy-preserving mobility data management, highlighting the design principles of a privacy-aware trajectory query engine and the types of attacks to user privacy that such an engine should be able to block. Section 11.3 discusses Hermes++ putting emphasis on the auditing and the fake trajectory generation algorithms that are implemented as part of the query engine to support its functionality. Section 11.4 demonstrates the Private-Hermes benchmark framework. In Sect. 11.5, HipStream privacy-preserving system is presented. Section 11.6 summarizes this chapter.

11.2 Background

Research in the domain of privacy-preserving data publishing has progressed along two main directions: providing off-site publication of sanitized data and providing on-site, restricted access to in-house data.

The first 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, 2, 13, 16, 27, 29, 30].

Hoh and Gruteser [9] 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 [30] 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. Nergiz et al. [16] also rely on the sequential nature of mobility data and propose a coarsening strategy to generate a sanitized dataset that consists of k-anonymous [27, 29] 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. In order to achieve space-time translation, the authors proposed W4M [2], which uses a different distance measure that allows time-warping.

In the second category, methodologies have been proposed for disclosure control in statistical databases [3]. These approaches protect sensitive information in a database while allowing statistical queries such as count and/or sum queries but no other information can be made available to the inquirer. According to the authors, addressing privacy violation problems can be classified into four main categories: (1) conceptual, (2) query restriction, (3) data perturbation, and (4) output perturbation. In the conceptual approach, two different data models are included. The conceptual model explores the privacy problem at the conceptual level while the lattice model comprises a framework for data represented in tabular form. Query restriction approach provides answer either by restricting the size of the set of the query or by controlling the overlap between successive queries. In the third category, attacks can be handled through data perturbation. Queries are answered according to a perturbed database. Essentially, a set of alteration / modification methodologies is used aiming for the best possible result w.r.t. privacy-preservation and data utility. Contrary, in the output perturbation approach the answer of the query is computed and then noise is added to the answer.

A privacy-aware query engine, as a protection mechanism, was first envisioned by Gkoulalas-Divanis and Verykios [8] (Fig. 11.1). The design principles of a query engine that protects users' privacy by generating fake trajectories are described. The idea behind that 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

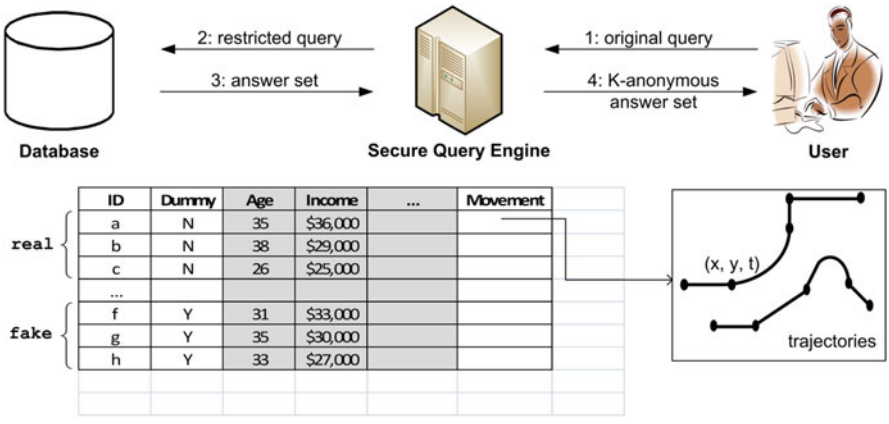

Fig. 11.1 A big picture of the system architecture [8]

data to support their analytic tasks. The engine allows subscribed end-users to gain restricted access to the trajectory data in order to perform various analysis tasks while preserving users' privacy from several types of attacks. It supports range, distance, k-nearest neighbor, landmark, route and queries for aggregative statistics for both trajectory (movement) and non-trajectory (relational) data. When a user poses a query, the engine retrieves the real trajectories r that belong to the answer set and combine them with $k - r$ fake trajectories in order to maintain k-anonymity principle by ensuring that the malevolent is not able to distinguish the real trajectories with high confidence. The necessary fake trajectories are generated based on an interpolation technique applied on pairs of real trajectories without though taking into consideration the time dimension.

Regarding the attacks that malevolent users may try to pursue in the original database, they are classified in three types:

- *User identification attack*: the identity of the user can be exposed by ad-hoc queries involving overlapping spatiotemporal regions.
- *Sensitive location tracking 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). Such locations are called sensitive for the user as they should not be disclosed to the attackers.
- *Sequential tracking 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 she has visited.

In the section that follows, we present Hermes++, a privacy-aware query engine, and we pay particular attention to the specific procedures it performs in order to block these types of attacks.

11.3 Hermes++ Query Engine

In this section, we present the architecture of Hermes++ query engine proposed by Pelekis et al. [22] and the algorithms that deliver its functionality. In particular, Sect. 11.3.1 provides details about Hermes++ architecture, Sect. 11.3.2 describes the algorithm that generates realistic fake trajectories, and Sect. 11.3.3 presents the auditing technique that is used to audit user queries and preserve the privacy in the answers to the queries.

11.3.1 Hermes++ Architecture

Hermes++ exploits on the trajectory storage functionality and the spatiotemporal query processing capabilities of Hermes for providing privacy-aware queries to

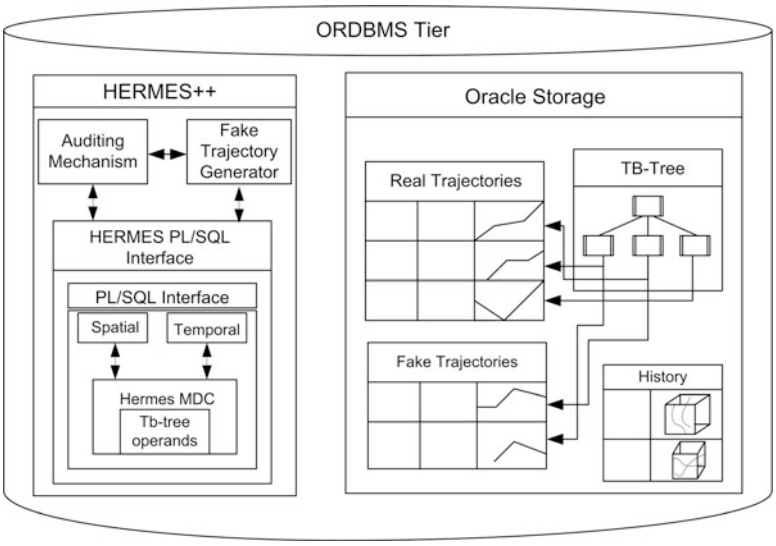

Fig. 11.2 The architecture of Hermes++ [22]

end-users. More specifically, Hermes defines a trajectory data type and a collection of operations as an Oracle data cartridge, which is further enhanced by the TB-tree access method [26] for efficient querying on trajectory data. Hermes++ directly utilizes this functionality at the ORDBMS level to store fake trajectories, as well as any historical 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 the embedded auditing module, which invokes the Hermes queries and the fake trajectory generator algorithm. Since the entire framework is built at the ORDBMS level, end-users are also able to pose their queries through PL/SQL (i.e. not only via the GUI). 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 11.2 illustrates the Hermes++ architectural framework.

As observed in this architecture, the two key components of Hermes++ functionality are the fake trajectory generator and the auditing mechanism (see the top left part of the architecture). These components are crucial for Hermes++ performance and will be described in detail in the sections that follow.

11.3.2 Fake Trajectory Generation

The Fake Trajectory Generation algorithm, originally presented in [22], aims 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 the privacy-aware query

mechanism. When a user poses a query to the database, the engine provides the answer only if at least L real user trajectories exist in the area. Lower bounding 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 Sect. 11.3.3) 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 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 the auditing mechanism (to be presented in Sect. 11.3.3) to handle different types of attacks from malevolent users.

The fake trajectory generation algorithm is based on the idea of the *Representative Trajectory Generation* (*RTG* for short) algorithm, introduced by Lee et al. [12]. 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, the original *RTG* algorithm fails to consider the temporal dimension of the generated trajectory. Therefore, fake trajectory generation algorithm transforms the *RTG* output by appropriately integrating the time dimension into the fake trajectory generation process.

Algorithm 11.1 provides the details of the fake trajectory generation approach. The algorithm takes as an input a set of line segments S_i resulting from a set of trajectories which form the answer to a user query. In the first step (line 1), the representative trajectory is produced based on this set of line segments of trajectories. For simplicity, in Fig. 11.3 segments are depicted as consecutive parts

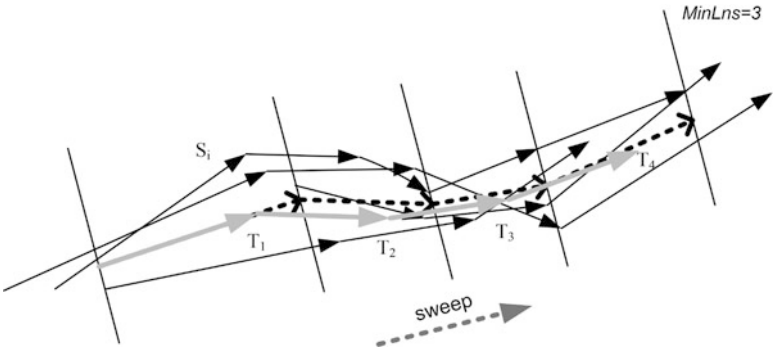

Fig. 11.3 Generating a fake trajectory over a set of line segments [22]

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 *RTG* assumes that all segments follow the same directional pattern. Then, *RTG* 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 Fig. 11.3.

After calculating the representative trajectory, the algorithm inserts the time dimension to each line segment and performs additional computations to adjust it and make it more plausible. In detail, a realistic length and speed for the 3D segments of the fake trajectory are examined and required. In Fig. 11.3, 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 2). 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 3) as a function of the initial timestamp t_0 and the duration of the fake trajectory (i.e., $|fake\_trajectory| * T$). If $(t_{max} > t_{MBBmax})$ then a line simplification procedure is applied to reduce the number of line segments (lines 5–9). Douglas-Peucker algoriothm [6] 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 (line 6).

The compression procedure is repeated until $t_{max} < t_{MBBmax}$ and at each iteration 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 10–13) 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 14) 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 $0 \leq i < |fake\_trajectory|$).

Algorithm 11.1 Fake trajectory generation

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}, $avgU_{min}$, $avgU_{max}$

1: *fake _trajectory* ← *RTG* (S_i, *MinLns,* γ)
2: calculate initial timestamp t_0 of the fake trajectory
3: $t_{max} \leftarrow t_0 + |fake\_trajectory| * Timestep$
4: **if** $t_{max} > t_{MBBmax}$ **then**
5: **repeat**
6 *Douglas _Peucker(fake_trajectory, f)*
7: $f \leftarrow f/2$
8: $t_{max} \leftarrow t_0 + |fake\_trajectory| * Timestep$
9: **until** $t_{max} < t_{MBBmax}$
10: **if** $l_{max} > 2 * l_{avg}$ **then**
11: $l_{max} \leftarrow random\ (l_{avg}, 2 * l_{avg})$
12: **else**
13: $l_{max} \leftarrow l_{avg} * random\ (1,\ l_{max}/l_{avg})$
14: **for each** $p_i \in fake\_trajectory$ **do**
15: set timestamps of the initial and final point of p_i
16: calculate speed U_i of p_i
17: **if** $U_i < avgU_{min}$ or $U_i > avgU_{max}$ **then**
18: **repeat**
19: $l \leftarrow random\ (l_{min}, l_{max})$
20: calculate new speed U_i of p_i
21: **until** $U_{min} < U_i < U_{max}$
22: calculate angle φ_i
23: define coords of new ending point based on l
24: map match *fake_trajectory*
25: **return** *fake_trajectory*

After assigning the time dimension to the current segment p_i (line 15), the algorithm proceeds to calculate the speed U_i for each segment p_i (line 16) and

checks if it lies within $avgU_{min}$ and $avgU_{max}$ (lines 17–21). 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 22–23).

Depending on the direction of the segment and its angle ϕ_i with x-axis, the fake trajectory generation algorithm calculates the new coordinates (x_{t+1}, y_{t+1}), according to the following formulas (l is the length of the line segment):

$$\phi_i = \arctan 2(y_{t+1} - y_t, x_{t+1} - x_t)$$

$$x_{t+1} = x_t + l * \cos(\phi), y_{t+1} = y_t + l * \sin(\phi)$$

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 (line 24) by employing a state-of-the-art map-matching algorithm [5]. This functionality of the algorithm can lead to a more realistic representation of the fake trajectory. 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 25).

11.3.3 Query Auditing

The main goal of Hermes++ query engine is to prevent the potential attacks that may occur while a malevolent user query the database. 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 the differences between the created fake trajectories which, in turn, increases her confidence regarding information about the corresponding real trajectories. To block this type of attack, Hermes++ uses auditing to track the queries initiated by each end-user in the system and denies answering overlapping queries.

Sensitive location tracking attack allows malevolent users to learn sensitive locations that real users have visited, and (possibly) reveal the identity of these users. To block these attacks, Hermes++ protects 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, illustrated in Fig. 11.4. 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 by collecting more precise information about the real trajectories on every focused query, which in turn increases her confidence. To block the sensitive location tracking attack, the auditing approach

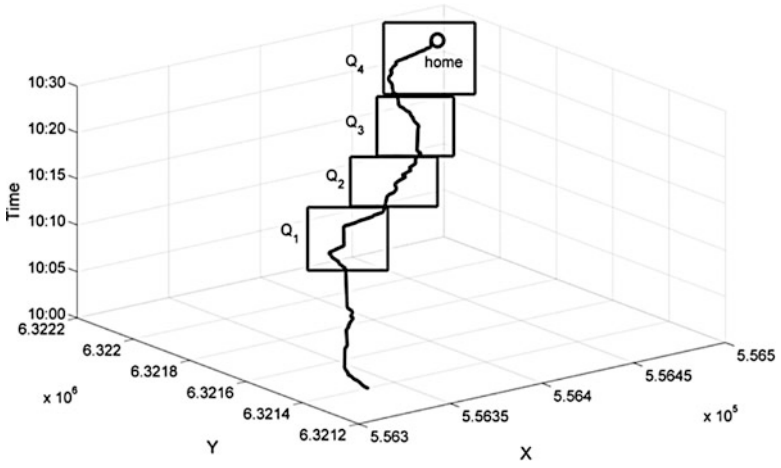

Fig. 11.4 Sensitive location tracking and sequential tracking attacks to user privacy [22]

identifies sensitive locations of trajectories that appear in the query window and proceeds to dislocate them so that the sensitive location is not disclosed.

Finally, 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 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.

The query auditing approach for shielding the database against malevolent users (to be presented in Algorithm 11.3) is based on the *Hide Sensitive Location Algorithm*, originally presented in [22], that is discussed first. This algorithm (listed in Algorithm 11.2) 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 1). For each trajectory of the given set *T*, it defines those sensitive locations, SL'_i, that correspond to the current trajectory (lines 2–3). 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 4–6). Otherwise, it computes a new synthetic (fake) trajectory that is then stored for future reference (lines 8–13).

Algorithm 11.2 produces fake (synthetic) sub-trajectories by applying a variant of the GSTD trajectory synthesizer, called GSTD*, proposed by Pelekis et al. [24]. 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 GSTD* 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 completed 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.

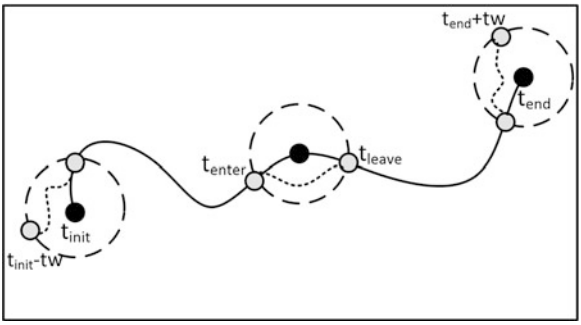

Fig. 11.5 Protecting sensitive locations of user trajectories [22]

The idea of hiding sensitive locations of a trajectory by misplacing its route is illustrated in Fig. 11.5. 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 8) (see the filled gray circles in Fig. 11.5). 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 (lines 9–10). 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 11). The algorithm returns the set of trajectories that does not any longer contain sensitive locations (line 14).

Algorithm 11.2 Hide sensitive locations

function HideSensitiveLocations(set of sensitive locations *SL*, set of trajectories *T*, user's query *MBB*, temporal window *tw*)

1: $SL' \leftarrow SL$ inside *MBB*
2: **for each** ($T_i \in T$) **do**
3: $SL'_i \leftarrow$select the subset of SL' that correspond to T_i
4: **for each** (sensitive location of T_i, $SL'_{i,j} \in SL'_i$) **do**
5: **if** (fake sub-trajectory computed in the past for this $SL'_{i,j}$) **then**
6: Retrieve the fake sub-trajectory from *History* and update T_i
7: **else**
8: $focal_{points} \leftarrow$ Intersection(T_i, buffer($SL'_{i,j}$, tw))
9: **if** ($|focal_{points}| = 1$) **then**
10: $focal_{points} \leftarrow$AddRandomPointOnSurface(buffer($SL'_{i,j}$, tw))
11: Produce a fake trajectory by applying *GSTD** on $focal_{points}$
12: Update the part of T_i with the fake sub-trajectory
13: UpdateHistory
14: **return** (T)

Algorithm 11.3, originally presented in [22], describes the query auditing mechanism. 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 serving the query (lines 1–2) 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 3). 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, the *Hide Sensitive Location Algorithm presented* earlier is invoked (line 4).

Having protected the sensitive locations of the trajectories in the querying region, Algorithm 11.3 commands the generation of the necessary fake trajectories for this region (lines 11–21). To generate the requested number of fake trajectories, the algorithm calculates a set of basic statistics (line 11) that are needed by the fake trajectory generation approach (Algorithm 11.1), while trying to find trajectories that follow more or less the same direction in the query region (lines 12–20). Specifically, a step dir_{step} (in degrees) is randomly selected (line 13) in the range 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 Fig. 11.6, 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 11.1 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° range is exceeded. Then, the algorithm selects a new dir_{step} and repeats the same process, until the requested number of fake trajectories is generated (line 21). 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 11.3 takes the necessary measures to protect the privacy of the users whose movement is depicted in the

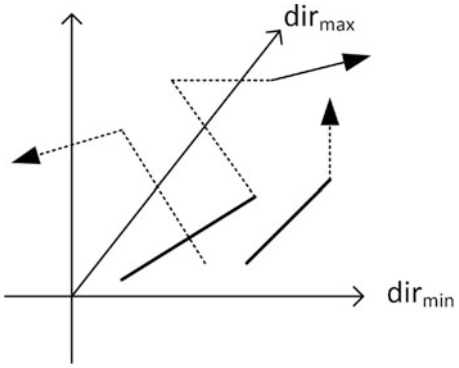

Fig. 11.6 Selecting segments from real trajectories [22]

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 23) 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 Fig. 11.7a, 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.

Algorithm 11.3 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*)

1: **if** CheckHistory(user posed in the past an overlapping query w.r.t. *MBB*) = true **then**
2: **Privacy threat:** Overlapping queries
3: $TR \leftarrow$SpatioTemporalRangeQuery(*MBB*)
4: $TR \leftarrow$ HideSensitiveLocations(*SL, TR, MBB, tw*)
5: **if** (CheckHistory(user posed in the past a nearby query w.r.t. s_{thr}, t_{thr}) = true) **then**
6: **Privacy threat:** Sequential tracking attack
7: **else**
8: **if** $|TR| \leq L$) **then**
9: **Privacy threat:** Lower bound threshold violation
10: **else**
11: CalculateStatistics (d_{min}, $d_{max}, l_{min}, l_{max}$, $l_{avg}, avgU_{min}, avgU_{max}$)
12: **repeat**
13: $dir_{step} \leftarrow random(0, dir_{step_{max}})$
14: $dir_{min} = random(0,360)$; $dir_{max} = dir_{min} + dir_{step}$
15: **repeat**
16: $S_i \leftarrow$ FilterbyDirection(dir_{min}, dir_{max}, TR)
17: $FT \leftarrow FT \cup$ Fake_Gen(S_i, *MinLns*, γ, *Timestep, MBB, Statistics)*
18: $dir_{min} \leftarrow dir_{min} + dir_{step}$
19: $dir_{max} \leftarrow dir_{max} + dir_{step}$
20: **until** $dir_{max} > 360$
21: **until** $|FT| = N$
22: Retrieve from *History* all fakes FT_{hist} from a nearby query of the user w.r.t. s_{thr}, t_{thr}
23: $P_{match} \leftarrow$ MinRandomMax(FT, FT_{hist})
24: **for each pair** $P_i\ (T_j, T_k) \in P_{match}$ **do**
25: **if** (*MBB* touches a historic query of the user) **then**
26: SpaceTimeTranslation(P_i)
27: **else**
28: $focal_{points} \leftarrow (T_{j_{end}}, T_{k_{start}})$
29: GSTD* ($focal_{points}$)
30: Update in FT the fake trajectory that corresponds to P_i
31: $FT \leftarrow$ HideSensitiveLocations(*SL, FT, MBB, tw*)
32: UpdateHistory
33: **return** ($TR \cup FT$)

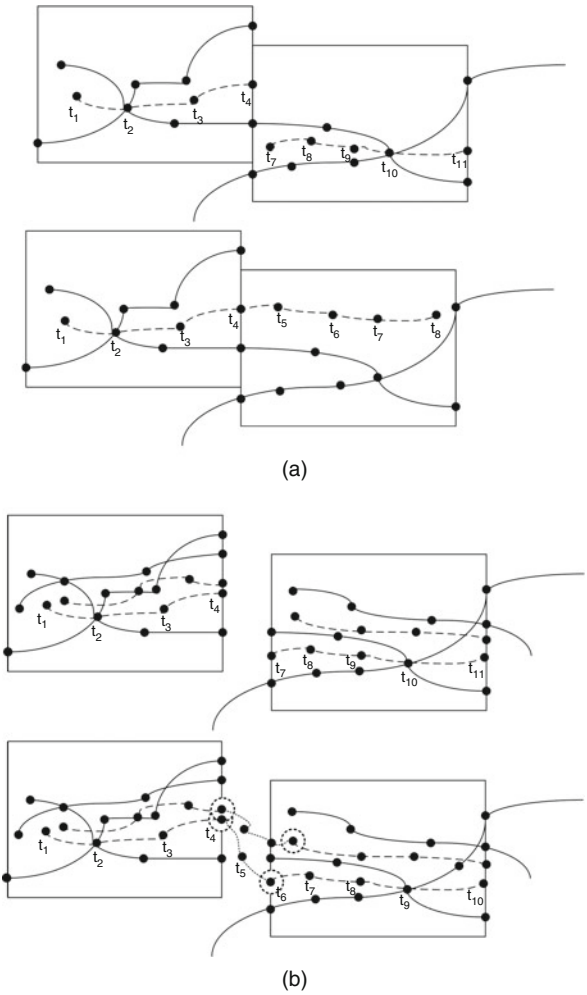

Fig. 11.7 Prohibiting sequential tracking: (**a**) case I, (**b**) case II [22]

Then, the algorithm checks the time dimension to assure that 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 (illustrated in Fig. 11.7b), where a spatial and/or a temporal gap exists between *MBB* and MBB_{hist}, Algorithm 11.3 generates a connection-trajectory (see the dotted lines) between them using GSTD*. 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 31), to conceal the fact that they are fakes.

As a last remark, *TrajAuditor* (Algorithm 11.3) 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 *TrajAuditor*. *TrajFaker* differs from *TrajAuditor* in the following steps. When a user executes a query, *TrajFaker* finds the trajectories that are contained in the specific spatiotemporal window (or the k nearest neighbors in case of $k-NN$ 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 11.1 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 execute overlapping or sequential queries, the fakes will appear in all of these queries' answers.

11.4 Private-Hermes Benchmark Framework

Building on top of Hermes++, Private-Hermes, developed by Pelekis et al. [23], integrates algorithms that enable the privacy-aware publishing of personal mobility data under a common, benchmark-oriented framework and gives the ability to users to evaluate the utility either of the fake or the sanitized trajectories via a variety of well-known mobility data mining algorithms, i.e. various types of clustering, frequent sequential patterns, etc. The idea is that by adding fake trajectories (that affect the cardinality of the MOD), as well as perturbating original ones (that affects the shape of the MOD) should not destroy the patterns hidden in the original MOD. Such an evaluation can be done by using clustering and frequent pattern mining techniques, appropriate for mobility data. Private-Hermes incorporates the following state-of-the-art algorithms:

- *Clustering*: Private-Hermes supports TRACLUS [12], T-Optics [14], and CenTR-I-FCM [24]. Two traditional clustering techniques, namely K-medoids [11] and Bisecting K-medoids [28], are also included with the special feature that the user can choose different distance functions between the trajectories (i.e. grouping only by their starting or destination point, without taking into account the whole route) [20].
- *Trajectory representatives*: related to cluster analysis, a useful requirement is to extract a compact representation of a set of trajectories (e.g. a cluster found through cluster analysis), in terms of "representative" trajectory. To this end, Private-Hermes supports CenTra "centroid" trajectories [24] and TRACLUS "typical" trajectories [12].

- *Frequent pattern mining*: Private-Hermes incorporates the T-pattern mining technique [7], which models sequences of visited regions, frequently visited in the specified order with similar transition times, out of trajectory databases.
- *Sampling*: Private-Hermes supports a state-of-the-art trajectory sampling technique proposed in [25].
- *Trajectory anonymization*: Private-Hermes incorporates NWA [1] and W4M [2].

The above-presented functionality is integrated in the Hermes MOD engine [21] by appropriately extending the query language with new constructs, in a fashion origi-nally proposed by Ortale et al. [19]. This allows users to progressively analyze the MOD and interchange between querying and mining operations. In detail, Pivate-Hermes users are given the ability to perform:

- *Querying and mining operations on Hermes*: the platform is capable of executing range and $k - NN$ queries on Hermes as well as mining operations using the algorithms listed above. Queries and mining operations are posed via Private-Hermes GUI, which provides essential capabilities, including query predicate selection, parameters selection and results projection. Graphical map user-interaction for predicate definition is also supported.
- *Privacy-aware querying on Hermes++*: users are able to run range and $k - NN$ queries enabling Hermes++, which protects from privacy attacks. The data owner requires that at least a certain number of trajectories are returned to the end-users in response to their queries, for all different types of supported queries. The result consists of a set of carefully crafted, realistic fake trajectories aiming to preserve the trend of the original user trajectories.
- *Comparison/evaluation of anonymization algorithms*: as already mentioned, Private-Hermes integrates NWA and W4M anonymization algorithms. Both algorithms take as input trajectories which may have been extracted from a query posed to Hermes, and transform them into anonymous equivalents, subsequently stored in the MOD. An advantage of the platform is its ability to design and execute benchmarks that evaluate the results from the application of anonymization algorithms regarding the distortion over real user trajectories. The incorporated data mining techniques can be applied, and patterns steaming from original data with patterns resulting from anonymized data can be compared. This can be achieved by executing queries in the original and the anonymized data (or patterns), and comparing the results.
- *Profiling end-user's behavior to identify malevolent users*: The platform supports query auditing techniques [8], which can be used to monitor the behavior of the end-users and build user profiles. These user profiles can be subsequently analyzed by the data owner, as explained in [8], to help her identify suspicious behavior of end-users in the system.

Figures 11.8 and 11.9 illustrate representative snapshots of Private-Hermes GUI. More specifically, in Fig. 11.8a, a dataset has been extracted using a range query, while in Fig. 11.8b the dataset has been anonymized using NWA [1]. From these

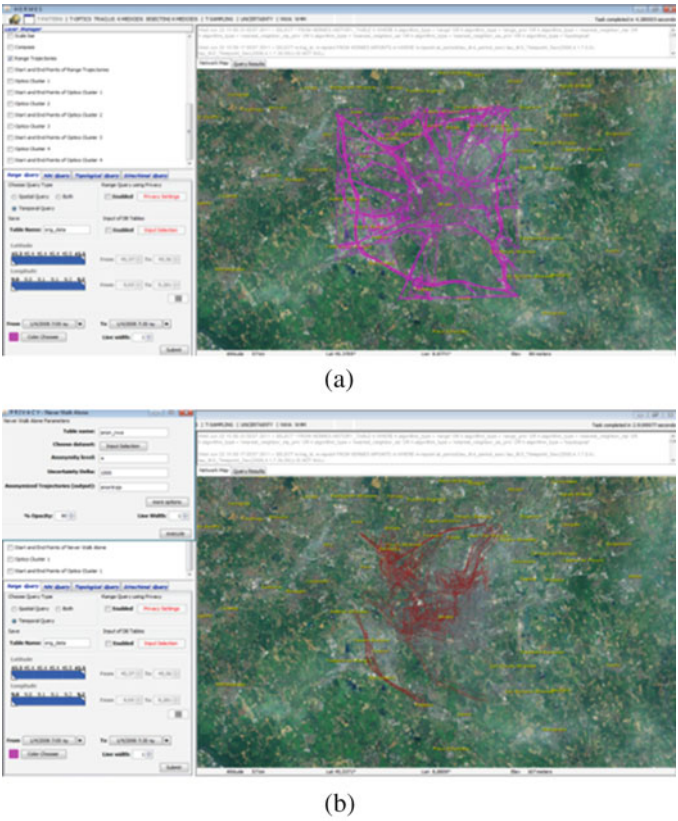

Fig. 11.8 The result of a range query in its (**a**) original vs. (**b**) NWA anonymized version [23]

outputs, a user can compare the distortion that has been caused to the dataset due to the anonymization algorithm. As a progressive analysis, Fig. 11.9a illustrates the result from the application of T-Optics [14] clustering on the original dataset (i.e. the one illustrated in Fig. 11.8a) in comparison with Fig. 11.9b, which presents the respective result when T-Optics is applied on the anonymized dataset (i.e. the one illustrated in Fig. 11.8b).

As for the technicalities of Private-Hermes components, illustrated in Fig. 11.10, the user interacts with a GUI with 3D rendering capabilities developed in Java and based on the Swing GUI widget toolkit [18]. The results from the operations that the program supports are visualized in the 3D globe provided by NASA World Wind [15]. To draw the charts reporting performance results, the JFreeChart library is

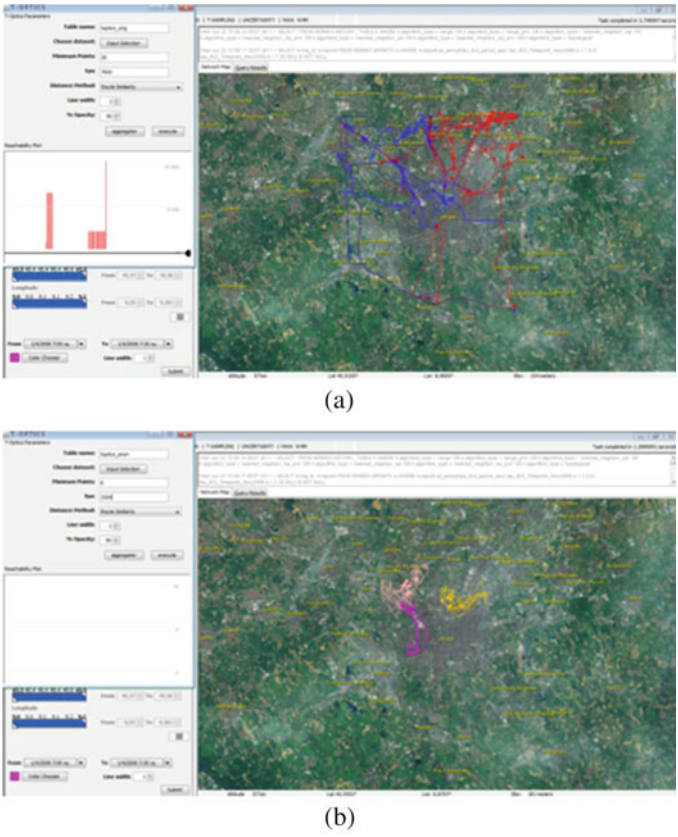

Fig. 11.9 T-Optics applied on (**a**) the original vs. (**b**) the anonymized dataset [23]

used [17]. Every component and library used during the development process is open source. Through the provided GUI, the user is able to setup his/her benchmark or, more generally, his/her analysis scenario. Private-Hermes retrieves the necessary data by calling the Hermes MOD engine.

The supported mobility data mining and anonymization algorithms have been incorporated as modules of the extensible DAEDALUS's MO-DMQL [19], while both of these sets of algorithms exchange data (i.e. real/fake/anonymized trajectories and mining models) directly with the database layer.

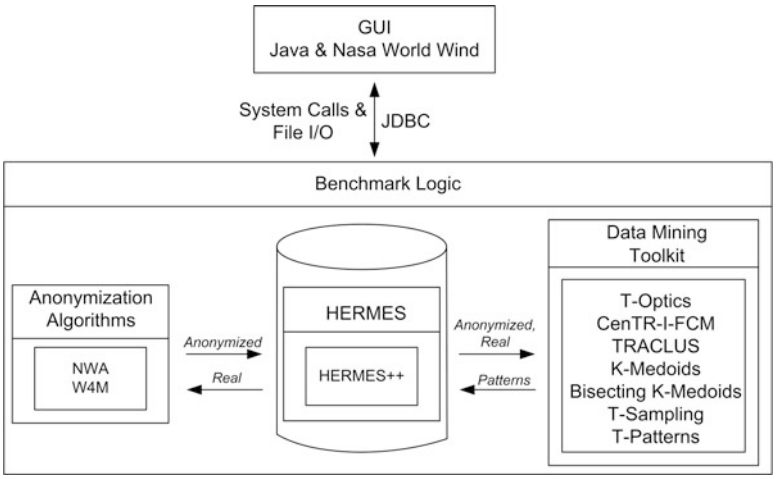

Fig. 11.10 Private-Hermes architecture [23]

11.5 HipStream: A Privacy-Preserving System for Managing Mobility Data Streams

The ability to build database systems able to connect individuals' information across different data repositories turns out to be simpler since individual information become more ubiquitous. Consequently, the privacy provisions incorporated into data collections and the privacy regulations that shield personal data, are debilitated. Data owners have no guarantee that the data that are donating are not misused for the sake of knowledge extraction since they have no control whether the privacy policies are enforced or not.

In order to deal with the lack of systemic control over the data use, the concept of Hippocratic management system was introduced by Agrawal et al. [4] to guarantee privacy and security of information they manage as a founding principle. Hippocratic databases extend the functionalities of traditional databases with privacy-preserving capabilities. The goal is to prevent disclosure of private information by placing data donor privacy as a main concern throughout data collection and management. Ten fundamental principles have been proposed that guide the behavior of Hippocratic stream data management:

- *Purpose Specification:* description of the purpose for which the data is collected needs to be collected and associated with the data itself.
- *Consent:* the purpose for which the data is collected has the consent of the user.
- *Limited collection:* collect the minimum amount of data from a user that satisfies the user's specified purposes
- *Limited use:* the purposes of the collected data should not be violated by operations carried out.
- *Limited Disclosure:* no personal information should be released to third parties without data owner's permission.

- *Limited Retention:* after the purpose of data collection is satisfied, user's data should be directly deleted.
- *Accuracy:* the information stored in the database system should be accurate and up-to-date.
- *Safety:* the adoption of security measures for protecting sensitive data from various types of attack.
- *Openness:* the individual whose data are recorded is allowed to access all information that is stored in the database and is related to herself.
- *Compliance:* data owners are able to validate that the privacy principles are conformed.

Wu et al. [31] developed a data management system, the so-called HipStream, which implements some of the aforementioned Hippocratic principles such as limited collection, limited use and limited disclosure. Data streams are collected and dropped dynamically in a system according to the data owner's policy. When data tuples arrive, the system is responsible to decide whether the data should be collected to serve the query or stored for analysis purposes in order to achieve limited collection. Controlling the access to the data w.r.t. data provider's preferences leads to limited disclosure. HipStream is able to preserve the privacy of the data streams that are shared between data providers and data users. The system guarantees not only that data providers' defined privacy specifications are enforced but also that the access to the data is limited. The idea behind HipStream is that service providers are allowed to access part of the data streams which are entirely controlled by data providers.

The architecture of HipStream is illustrated in Fig. 11.11. The basic components of the system are Security Manager, Privacy Controller, Query Management and Stream Manager. Through Web interface each end user has the ability to generate, retrieve and manage protected stream data.

A data owner is first registered to the system and specifies her privacy preferences such as who, for what purpose, under what conditions and which parts. Privacy policies are designed based on modelling users, data to be accessed and data accessing purposes with hierarchical categories. The preferences are then registered into the Policy Controller which is responsible for maintaining the privacy.

Prior to the registration of a data stream to the system, the Stream Registration acquires the purpose of the stream data and the consent of the stream owner. The Stream Manager receives the input stream and prepares it for further processing inside the system.

On the other hand, service providers may pose queries directly to the system while defining at the same time the query purpose. The query is forwarded through query getaway to the query rewriter. Query rewriter is then responsible to examine if the privacy preferences are satisfied. In case where the policies are not met, the query is rewritten. The query is dropped if its purpose is not in line with the authorised purpose on using the data. At this level, limited disclosure and limited use principles are enforced.

Next, the query is forwarded to the query manager and the stream filter. The stream filter is in charge of enforcing limited collection principle. The records that

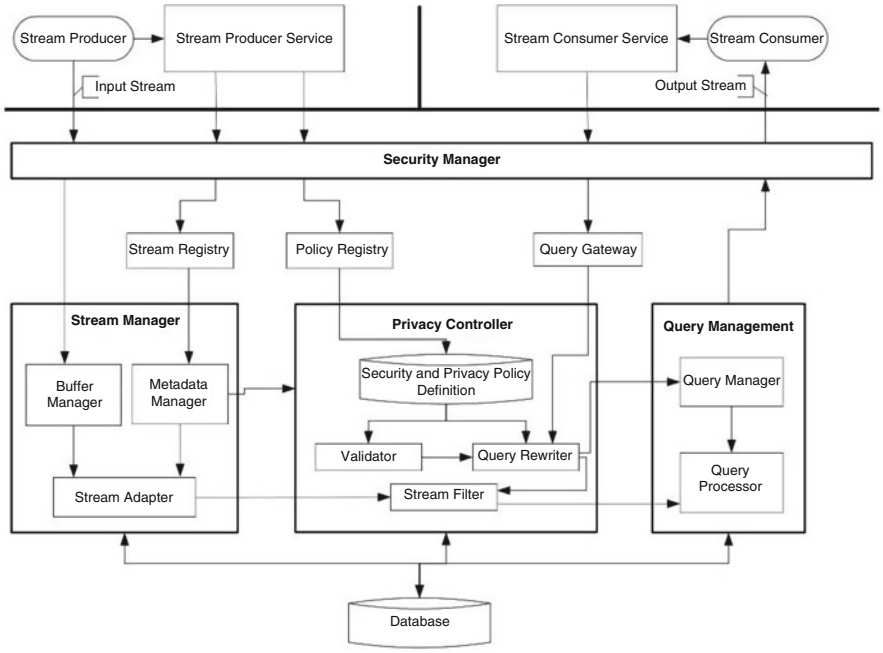

Fig. 11.11 HipStream architecture [31]

are participating to the answer set are maintained while the others are dropped. Moreover, the attributes that are not asked by any query are anonymized (i.e. replaced by null). Query processor executes the registered continuous queries and streams the result out to the query owner.

11.6 Conclusions

In this chapter, we presented techniques and Mobility Data Management Systems that have been proposed in the literature able to preserve the privacy of the users whose data are kept to the hosting organisation for analysis purposes. Hermes++ is a privacy-aware query engine that enables the remote analysis of user mobility data, 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. On top of Hermes++, Private-Hermes is an integrated platform for applying data mining and privacy-preserving querying over mobility data. Finally, Hipstream, a data stream management system aiming at preserving users' privacy by enforcing Hippocratic principles was presented. Limited collection, limited use and limited disclosure of data are the main privacy requirements that the system implements.

References

1. Abul, O., Bonchi, F., and Nanni, M. (2008) Never walk alone: Uncertainty for anonymity in moving objects databases. In Proceedings of ICDE, pages 376–385.
2. Abul, O., Bonchi, F., and Nanni, M. (2010) Anonymization of moving objects databases by clustering and perturbation. Information Systems, 35(8), pages 884–910.
3. Adam, N.R. and Worthmann, J. C. (1989) Security-control methods for statistical databases: A comparative study. ACM Computing Surveys, 21(4), pages 515–556.
4. Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002) Hippocratic databases. In Proceedings of VLDB Endowment, pages 143–154
5. Brakatsoulas, S., Pfoser, D., Salas, R. and Wenk, C. (2005) On map-matching vehicle tracking data. In Proceedings of VLDB Endowment, pages 853–864.
6. Douglas, D. and Peucker, T. (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization, 10(2), pages 112–122.
7. Giannotti, F., Nanni, M., Pedreschi, D., and Pinelli, F. (2007) Trajectory Pattern Mining. In Proceedings of SIGKDD, pages 330–339.
8. Gkoulalas-Divanis, A. and Verykios, V. S. (2008) A privacy–aware trajectory tracking query engine. ACM SIGKDD Explorations Newsletter, 10(1), pages 40–49.
9. Hoh, B. and Gruteser, M. (2005) Protecting location privacy through path confusion. In SECURECOMM, pages 194–205.
10. Hoh, B., Gruteser, M., Xiong, H. and Alrabady, A. (2007) Preserving privacy in GPS traces via uncertainty-aware path cloaking. In Proceedings of CCS, pages 161–171.
11. Kaufman, L., Rousseeuw, P. J. (1990) Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, NY, Vol. 334.
12. Lee, J. G., Han, J., and Whang, K. Y. (2007) Trajectory clustering: a partition-and-group framework. In Proceedings of SIGMOD, pages 593–604.
13. LeFevre, K., DeWitt, D. and Ramakrishnan, R. (2006) Mondrian multidimensional *k*-anonymity. In Proceedings of ICDE, page 25.
14. Nanni, M. and Pedreschi, D. (2006) Time-focused clustering of trajectories of moving objects. Journal of Intelligent Information Systems, 27(3), pages 267–289.
15. NASA, World Wind Java SDK. URL: http://worldwind.arc.nasa.gov/java. (accessed: 6 Oct. 2011).
16. Nergiz, M. E., Atzori, M. and Saygin, Y. (2008) Towards trajectory anonymization: A generalization-based approach. In Proceedings of the SIGSPATIAL, pages 52–61.
17. Object Refinery, the JFreeChart project. URL: http://www.jfree.org/jfreechart. (accessed: 6 Oct. 2011)
18. Oracle, The Swing Tutorial. URL: http://download.oracle.com/javase/tutorial/uiswing. (accessed: 6 Oct. 2011).
19. Ortale, R., Ritacco, E., Pelekis, N., Trasarti, R., Costa, G., Giannotti, F., Manco, G., Renso, C., and Theodoridis, Y. (2008) The DAEDALUS Framework: Progressive Querying and Mining of Movement Data. In Proceedings of ACM SIGSPATIAL, page 52.
20. Pelekis, N., Andrienko, G., Andrienko, N., Kopanakis, I., Marketos, G., and Theodoridis, Y. (2011) Visually Exploring Movement Data via Similarity-based Analysis. Journal of Intelligent Information Systems, 38(2), pages 343–391.
21. Pelekis, N., Frentzos, E., Giatrakos, N., and Theodoridis, Y. (2008) Hermes: Aggregative LBS via a trajectory DB engine. In Proceedings of SIGMOD, pages 1255–1258.
22. Pelekis, N., Gkoulalas-Divanis, A., Vodas, M., Kopanaki, D., and Theodoridis, Y. (2011). Privacy-aware querying over sensitive trajectory data. In Proceedings of CIKM, pages 895–904.
23. Pelekis, N., Gkoulalas-Divanis, A., Vodas, M., Plemenos, A., Kopanaki, D., and Theodoridis, Y. (2012) Private-Hermes: A Benchmark Framework for Privacy-Preserving Mobility Data Querying and Mining Methods. In Proceedings of EDBT, pages 598–601.

24. Pelekis, N., Kopanakis, I., Kotsifakos, E., Frentzos, E. and Theodoridis, Y. (2011) Clustering uncertain trajectories. Knowledge and Information Systems, 28(1), pages 117–147.
25. Pelekis, N., Panagiotakis, C., Kopanakis, I., and Theodoridis, Y. (2010) Unsupervised trajectory sampling. In Proceedings of ECML PKDD, pages 17–33.
26. Pfoser, D., Jensen, C. S., and Theodoridis, Y. (2000) Novel approaches to the indexing of moving object trajectories. In Proceedings of VLDB, pages 395–406.
27. Samarati, P. (2001) Protecting respondents' identities in microdata release. Transactions on Knowledge and Data Engineering, 13(6), pages 1010–1027.
28. Steinbach, M., Karypis, G., Kumar, V. (2000). A comparison of document clustering techniques. In Proceedings of KDD Workshop on Text Mining, 400(1), pages 525–526.
29. Sweeney, L. (2002) *k*-anonymity: A model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge Based Systems, 10(5), pages 557–570.
30. Terrovitis, M. and Mamoulis, N. (2008) Privacy preservation in the publication of trajectories. In Proceedings of Mobile Data Management, pages 65–72.
31. Wu, H., Xiang, S., Ng, W. S., Wu, W., & Xue, M. (2014) HipStream: A Privacy-Preserving System for Managing Mobility Data Streams. In Proceedings of Mobile Data Management, Vol. 1, pages 360–363.