A Privacy-Aware Trajectory Tracking Query Engine

Aris Gkoulalas-Divanis
Department of Comp. & Comm. Engineering
University of Thessaly
Volos, Greece
arisgd@inf.uth.gr

Vassilios S. Verykios
Department of Comp. & Comm. Engineering
University of Thessaly
Volos, Greece
verykios@inf.uth.gr

ABSTRACT

Advances in telecommunications and GPS sensors technology have made possible the collection of data like time series of locations, related to the movement of individuals. The analysis of this, so-called trajectory data, is beneficial both for the individuals (e.g., through location-based services) and for the community as a whole (e.g., decision support for urban planning or traffic control). However, because of the very nature of this data, strict safeguards must be enforced to ensure the privacy of the individuals, whose movement is recorded.

In this paper, we present a privacy-aware trajectory tracking query engine that offers strict guarantees about what can be observed by untrusted third parties. Through the query engine, subscribed users can gain restricted access to an in-house trajectory data warehouse, to perform certain analysis tasks. In addition to regular queries involving non-spatial non-temporal attributes, the engine supports a variety of spatiotemporal queries, including range queries, nearest neighbor queries and queries for aggregate statistics. The query results are augmented with fake trajectory data (dummies) to fulfil the requirements of K-anonymity. Through qualitative analysis, we prove the effectiveness of our approach towards blocking certain types of attacks, while minimally distorting the dataset.

1. INTRODUCTION

Recent advances in the technology of location-detection devices (e.g., GPS sensors, RFIDs, and cellular phones) made possible the collection of user location data at an accuracy of a few meters. Combined with the availability of simple interpolation techniques (e.g., [10]), it is nowadays possible to accurately approximate the actual user movement and reconstruct the corresponding user trajectory. Knowledge of user movement is essential to a wide variety of applications, ranging from car navigation systems and traffic control, to urban planing and location-based advertisements. However, potential disclosure of accurate trajectory information of an individual to untrusted third parties can severely compromise his or her privacy. For this reason, privacy preservation techniques are essential to control the accuracy at which trajectory information is communicated to untrusted entities.

In this paper, we consider a data holder, such as a governmental agency or a telecom operator, that collects movement information for a community of people. This information is used to approximate the real user movement and to create user trajectories, subsequently stored in a database. Apart from the analysis this data is undergone within the premises of the data holder, we assume that at least a portion of the data (if not all) has to become available to other, possibly untrusted, parties for analysis and querying purposes. As an example, consider the Octopus[1] smart RFID card, commonly used by Hong Kong residents both for transportation and for selected point-of-sale payments at convenience stores, supermarkets, on-street parking meters, etc. Currently, the Octopus company accumulates a large amount of movement data on a daily basis. Publication of this data would benefit several other parties, such as transportation engineers, insurance companies, and advertisers. For instance, advertisement companies would benefit by identifying locations (or routes) that are usually crowded at certain times during the day, in order to place advertisements for their products.

It is evident that direct publishing of this information, even after removing the obvious identifiers from the dataset (e.g., name, ID number) is insufficient to protect the privacy of the individuals, whose movement is recorded. Indeed, a malevolent user who wants to compromise the dataset (commonly referred to as a *snooper*) can still learn the movement habits of all users in the database and track them down to their house or place of work. A mechanism is thus needed to regulate the answer data that is made available to the inquirer as a result of his or her query. However, such a task is impossible to be accomplished for all types of possible queries, without severely sacrificing the quality of the data. Contrary to the research in privacy-aware data publishing [1; 2; 9], where authors make the implicit assumption that most of the knowledge in a dataset can become publicly available without causing privacy breaches, in our work we assume that the majority of the information that is stored in the database must remain private.

To protect user privacy, we use a query engine that restricts querying of the trajectory database to a set of pre-defined types of queries: (i) range queries (both in time and space), (ii) queries for landmarks (e.g., "all trajectories starting from A, ending at E and/or passing from B, C, D"), (iii) queries for given routes (e.g., "all trajectories crossing the Park street"), (iv) nearest neighbor queries, and (v) queries for aggregate statistics (e.g., "number of people visiting the city center from 8am to 10am"). Queries involving non-spatial non-temporal attributes are also supported. The engine retrieves real user data from the database and mixes it (if necessary) with carefully crafted dummy data. As a result, an honest user can still attain invaluable knowledge from the database, while this knowledge is insufficient for a snooper to identify or track down a specific subject. Our proposed system can block two types of attack:

User identification attack In this type of attack the identity of the user is exposed by ad hoc queries involving overlapping sets of spatial attributes. Each new query involves the attributes-values pairs of previous queries along with new pairs that

[1] Available at `http://www.octopuscards.com/`.

constitute it more specific.

Sequential tracking attack In this type of attack the user is tracked down through his or her trajectories by a set of focused queries on regions that either (partially) overlap or are adjacent to each other, in terms of space and time. Each new query attempts to "follow" the movement of the user by capturing a new part of the user trajectory.

In both types of attack, the malevolent user can increase his or her confidence regarding the identity of a user by segregating the dummies from the real user trajectories.

The rest of this paper is organized as follows. Section 2 presents the available related work. In Section 3 we provide our methodology for building the query engine and demonstrate its effectiveness towards tackling the two types of attacks. Section 4 details over the algorithms that are implemented as part of the query engine to accomplish the various tasks. In Section 5 we propose a query specification mechanism that relies on a spatial and a temporal hierarchy to ease the composition of complex queries. Finally, Section 6 provides a qualitative analysis of the proposed approach and Section 7 concludes this paper.

2. RELATED WORK

Privacy-aware data publishing has been the focus of attention for almost three decades. State-of-the-art research in data publishing is primarily conducted along two principal directions: *providing on-site, restricted access to in-house data*, and *providing off-site publication of sanitized data*. Methodologies in the first category provide mechanisms to secure databases against disclosure of confidential information. In their work, Nabil and Wortmann [3] survey the most prominent research on security control for statistical databases. Existing approaches are classified into four categories: *conceptual*, *query restriction*, *data perturbation*, and *output perturbation*. However, all the enforced security approaches are based solely on count and/or sum queries since no other information is made available to the inquirer. Our approach is an extension of the research conducted in this direction and especially in the subfield of query restriction. The query engine that we propose supports a large variety of queries, involving both trajectory and non-trajectory data.

Numerous methodologies for off-site publishing of sanitized data have been proposed over the years, spanning several research fields such as association rules hiding, classification rules hiding, data perturbation etc. Since our approach aims at concealing sensitive trajectory information, in what follows, we focus our attention on research in the field of trajectory data sanitization. Methodologies in this category aim at sanitizing datasets of trajectories, in such a way that in the outcome, the identity of the owner of each trajectory is properly shielded.

Hoh and Gruteser [5] present a path perturbation algorithm that produces fake traffic patterns to confuse an adversary with respect to the actual location of a user. The key idea of the algorithm is to identify locations where the paths of at least two users are close in proximity and to perturb the actual locations to create a fake "crossing" of the trajectories. The authors formulate the privacy problem as a constrained optimization problem and derive heuristics to efficiently solve it.

Abul, et. al [1] address the problem of hiding sensitive trajectory patterns in a database of moving objects. The authors propose a coarsening technique that involves the removal of some spatiotemporal points from trajectories supporting sensitive patterns, such that in the sanitized dataset these patterns are suppressed.

Terrovitis and Mamoulis [9] propose a data suppression technique to prevent adversaries that hold a projection of the data from inferring locations missing in their projection. The anonymization algorithm identifies points of threat and generalizes the original trajectories in such a way that they do not contain these points any longer. The algorithm simulates potential attacks from adversaries and protects against the identified privacy breaches, taking special care on minimally affecting the quality of the sanitized database.

Abul, et. al [2] propose a K-anonymization technique for trajectory data that exploits the inherent uncertainty regarding the whereabouts of the moving objects. The authors represent the uncertainty of a trajectory as a cylinder, consisting of a series of disks, defined for the different time intervals of movement. To enforce trajectory K-anonymity in the database, the authors consider a variant of the greedy clustering technique that co-clusters trajectories, while constraining the radius of the identified clusters.

3. METHODOLOGY

In what follows, we provide a big picture of the proposed system, along with the types of the supported queries and the implemented interpolation strategies. Furthermore, we shed light on the applied mechanism for the blocking of possible attacks. Specifically, in Section 3.1 we provide an overview of the system and its operation. Section 3.2 presents the different types of queries that are handled by the query engine, while Section 3.3 explains the interpolation techniques that are used for the generation of fake records. Finally, Section 3.4 provides an example of use and Section 3.5 presents the defense of the system against the two types of attack.

3.1 System overview

Consider a system that collects movement data for a population of users. Each user is equipped with a positioning device that regularly transmits his or her location (commonly referred to as a *location update*) to a trusted server. The server applies a trajectory reconstruction approach (such as [10]) to approximate the real route that the user followed from the last location update to the current one. Then, it updates the movement information of the user in the database. Apart from movement data, the system is also in possession of other information for the tracked users (e.g., name, age, income). Due to some mutual agreement, the data holder wants to provide access to his or her data to other interested parties. Thus, the data holder generates a view of the database that contains both the attributes and the tuples that he or she wants to release. For example, if access to the data is to be offered for an advertisement company, information about the income of people should be kept private. On the other hand, age information may be invaluable to allow for decisions regarding the types of products that will be advertised. Such a view of the database is presented in Figure 1. Each record of the database contains information for a single user, identified though a unique ID. The movement of the user is recorded as the collection of his or her trajectories and is stored in a special-type attribute of the record[2].

A user of the system has to register to the query engine to access and query the corresponding database view. Upon registration, the database table is augmented by a column (initialized to 'N' for all records) that holds information regarding the real and the fake trajectories. Furthermore, the data holder assigns a threshold K to the user that indicates the minimum number of records that should formulate the answer set of a user query. A set of procedures are made

[2]This way of storing trajectory data is in accordance to HERMES [7], a data cartridge for moving objects, implemented as an extension of the Oracle DBMS.

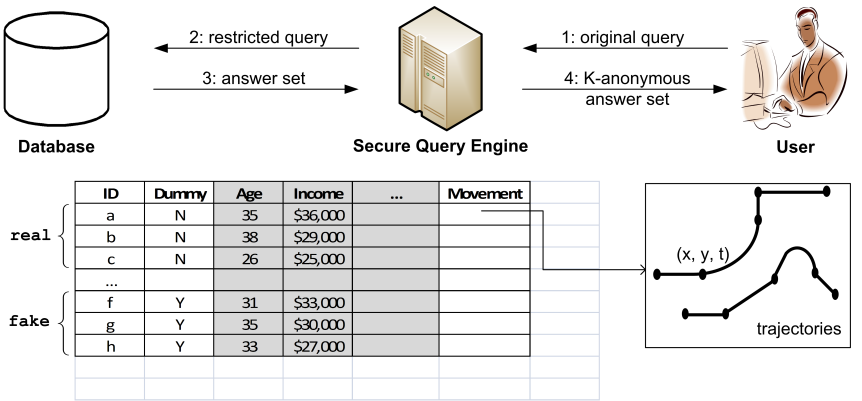


Figure 1: A big picture of the proposed system architecture.

available to the user to query the database (see Section 3.2). When a user submits a query to the query engine, the query is checked for conformance to a set of rules whose purpose is to shield the system against snoopers (see Section 3.5). In case of rule violation, the system denies execution of the query. On the other hand, if no rule is violated, the database is searched to identify the answer to the query. Count queries and queries referring to non-spatial non-temporal data (e.g., "find the average age of the recorded population") are handled by common strategies for query restriction and K-anonymity for relational data [3; 8]. However, in this paper, we focus our attention on spatial/temporal and spatiotemporal queries. Provided with such a query, the query engine identifies the records in the database that formulate the correct answer. If the size of the answer set is $R < K$, the system needs to produce $K - R$ fake records of dummy trajectories to ensure trajectory K-anonymity[3], unless these fake records have been already placed in the database from a previous application of the proposed technique. The answer set is formulated by the union of all the real records along with the fakes so as to reach a cardinality of K. The dummy trajectories are produced through interpolation applied on the real trajectories (see Section 3.3). Once produced, the dummy trajectories are maintained in the database along with the real user trajectories for coherence purposes; followup queries to the database should produce compatible answers with respect to past queries. On the other hand, if the number of retrieved records suffice to provide K-anonymity, the records are returned as is to the user, without the incorporation of any fakes. In every case, along with the answer to the query returned to the user, the system provides information regarding the percentage of the real records to the returned answer set.

3.2 Queries for trajectory data

The proposed query engine can handle queries for both trajectory and non-trajectory data. Queries for non-trajectory data include all types of queries that result either in count/aggregate data or in actual views of records that do not include the trajectory component (attribute). In the case of count queries, the system refrains from providing an answer that was computed by using less than K records. For example, a query for the number of users that have age under 16 will be serviced only if at least K users match this age limit. The same approach holds when querying for aggregate data (e.g., averages over non-trajectory data) or requesting records from the dataset that do not contain the trajectory component (e.g., "list all users having a yearly income above $15,000"). In all such cases, if K-anonymity cannot be satisfied, the query engine denies answering the query. In what follows, we focus our attention on the supported queries for trajectory data. Figure 2 summarizes them.

[3]By elevating the definition of K-anonymity [8] from relational to trajectory data, a user is K-anonymous if he or she can be matched to any among K different trajectories (commonly referred to as the *anonymity set*). In the best case scenario, given K trajectories, a user can be matched to the correct one with a probability of $1/K$.

3.2.1 Range and nearest neighbor queries

In range queries (see Figure 2(i)) the objective is to identify user trajectories that either (i) lie within a predefined spatial, temporal or spatiotemporal region, or (ii) are within a given distance d with respect to a point of reference. Here are some examples of range queries:

Q1 "Find all user trajectories that crossed the park".

Q2 "Find all user trajectories on Tuesday 1/1/2008, from 6am to 10am".

Q3 "Find all user trajectories that crossed the park this Monday, from 6am to 10am".

Q4 "Find all user trajectories that are at most 1Km away from the Computer Science department".

In k Nearest Neighbor (k-NN) queries, the objective is to identify the k nearest neighbors given a point of reference. An example of a k-NN query is:

Q5 "Find the five user trajectories that are closest to the park".

From the perspective of the query engine, all **Q1**-**Q5** queries are handled in the same manner. First, the answer set is computed. If it respects the requirements of K-anonymity then this answer set is returned to the user. If it fails to preserve K-anonymity (e.g., when $k < K$) a set of fake records is created to elevate the size of the answer set to K.

3.2.2 Landmark queries

In landmark queries (see Figure 2(ii)), the objective of the inquirer is to identify user trajectories that have a standard start/end point and/or pass through one or more given intermediate points. All these points are called *landmarks*. A landmark query can be either spatial or spatiotemporal. Examples follow:

Q6 "Find all user trajectories that start from the museum, move to the library and end at the national garden".

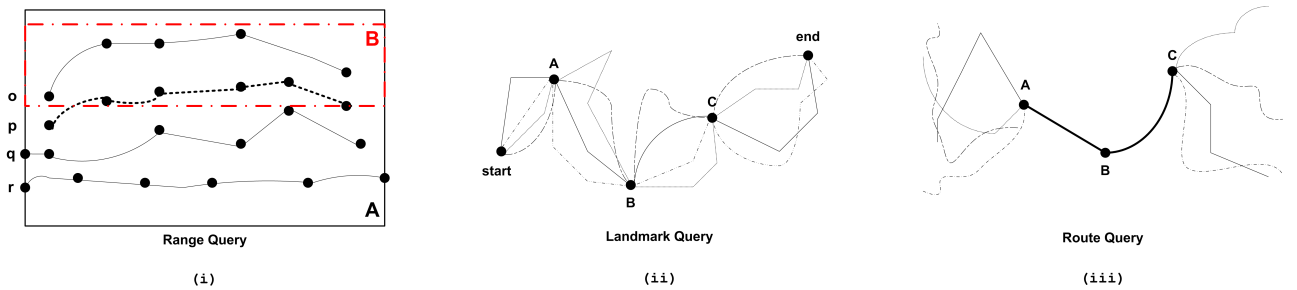


Figure 2: The supported types of queries when requesting trajectory data.

Q7 "Find all user trajectories that visited the cafè last Tuesday at 8am and then the US bank at 9pm".

In the case that at least one of the start/end landmarks is not specified, the user needs to supply a rectangle region that includes all landmarks and serves as the space within which the returned trajectories are depicted.

3.2.3 *Route queries*

In route queries (see Figure 2(iii)), the objective is to identify user trajectories that followed a common predetermined route. In contrast to landmark queries, in route queries all points in the route need to be part of the retrieved trajectories. Furthermore, the inquirer needs also to supply a rectangle region which contains the route and, as in the case of the landmark queries, lays out the area within which the returned trajectories should be depicted. Here are two examples of route queries:

Q8 "Find all user trajectories that entered the bridge at point A, continued to the highway at point B and left the highway at exit C".

Q9 "Find all user trajectories that entered the bridge (point A) at 8:30am today, continued to the highway (point B) and left the highway (exit C) at 8:40am".

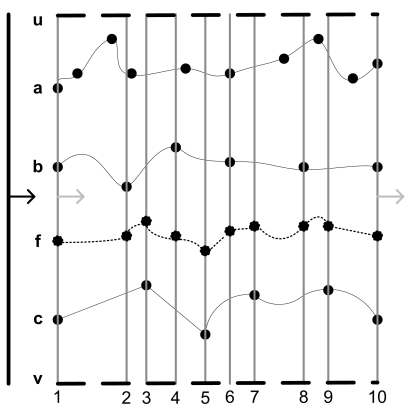


Figure 3: Interpolation of trajectory data.

3.3 Interpolation strategies

Fake records are created by applying interpolation techniques on the real data. In what follows, we consider two types of interpolation: numerical data interpolation and trajectory data interpolation. In both cases we assume that $K - R$ fake records have to be produced, where R is the number of real records in the answer set.

3.3.1 *Numerical data*

Numerical data interpolation involves the selection of appropriate values to fill in the numerical fields of the fake records. Selected values should lie close to the ones of the real records, while preserving the basic statistical properties of the answer set; that is the minimum, maximum and average. Suppose that we want to set the attribute values of a numerical attribute for the fake records. First, we generate $K - R$ empty records and we randomly partition them into pairs of two. Then, we calculate the mean m of the R records with respect to the considered numerical attribute and we identify the (signed) minimum d_{min} and the maximum d_{max} deviation of the R records from the mean. For each pair of fake records we randomly select two values $\pm d \in [d_{min}, d_{max}]$ and we set the attribute value of one record to $m - d$ and of the other record to $m + d$. Finally, if the number of fake records is odd, the attribute of the non-paired record is assigned to the mean m.

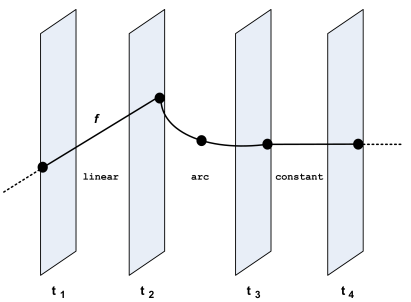


Figure 4: Basic movement types supported by HERMES.

3.3.2 *Trajectory data*

The interpolation of trajectory data proceeds in an iterative fashion. The proposed technique randomly selects at each step a pair of trajectories from the answer set. Then, for the selected trajectories, it sweeps the **x**-axis (see Figure 3) to collect their location updates in a sorted way. For each collected location update, the algorithm identifies the corresponding (x, y, t) point on the other trajectory that has the same **x**-coordinate. This is accomplished by utilizing knowledge regarding the initially applied trajectory reconstruction approach. For example, HERMES [7] supports the movement types presented in Figure 4. Having identified the two points, the algorithm computes a new point (x, y', t), where y' is the average of the two y-coordinates. By following the same process for all available location updates, coming from the two selected trajectories, a set of fake location updates is created for the dummy trajectory. Following that, the algorithm "reconstructs" the movement of the dummy trajectory by using the same reconstruction approach as the one of the real trajectories. For example, by using HERMES,

any of the available movement functions shown in Figure 4 will do. As a last step, the created dummy trajectory is inserted in the current answer set and the algorithm proceeds to the next iteration, until the answer set offers K-anonymity.

A slight enhancement is needed to the presented algorithm to handle the trajectories that are nearest to the border of the considered spatial/temporal or spatiotemporal area. If the algorithm is left as is, then these trajectories will always be real. To fix this issue, the algorithm considers the existence of two trajectories that represent constant movement and touch the $y = y_{min}$ and $y = y_{max}$ borders, respectively. These dummy trajectories will not be part of the answer set. However, their existence allows the algorithm to produce other dummies that will lie in-between the real nearest trajectories to the borders and the respective border. For example, in Figure 3, dummy trajectories $\mathbf{u}$ and $\mathbf{v}$ are used for this reason. The existence of $\mathbf{u}$ enables the generation of a dummy trajectory in-between $\mathbf{u}$ and $\mathbf{a}$. The same holds for trajectories $\mathbf{c}$ and $\mathbf{v}$.

3.4 An example of use

Having presented the elements of the proposed trajectory query engine, we proceed to present an example of its use. Consider the table of Figure 1, where the database holder requires $K = 6$ anonymity. Initially, it consists only of real trajectories. A user poses a query to the database, requesting all trajectories that lie within a given spatiotemporal region. Assume that from the whole database, only trajectories $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ satisfy the query. As a result, the query engine has to create three fake records as part of the answer set that will be returned to the inquirer. First, three records are created in the database, each having a unique ID and attribute `Dummy` set to `Y`. Second, for each numerical attribute, the query engine applies numerical interpolation to set its value. Attribute `Age` has a mean m of 33 and a deviation of $[min - m, max - m] = [-7, 5]$ around the mean. Since we have three fake records, one will be assigned the mean value (record $\mathbf{h}$) and for the other two we need to select two values $\pm d \in [-7, 5]$. Suppose that we select $d = \pm 2$. Then, one record will have `Age = 31` (record $\mathbf{f}$) and the other record `Age = 35` (record $\mathbf{g}$). The exact same process is followed for variable `Income`. The pairs of fake records are randomly formulated prior to each numerical attribute specification.

After assigning values to all numerical attributes, the query engine has to create dummy trajectories to be associated to the fake records. For clarity purposes, in Figure 3, we present the generation process for one dummy, considered to be assigned to record $\mathbf{f}$. We assume that at one iteration the trajectory interpolation algorithm chose trajectories $\mathbf{b}$ and $\mathbf{c}$ as the drivers for the generation of dummy trajectory $\mathbf{f}$. By sweeping the $\mathbf{x}$-axis, the location updates from these two trajectories are collected in the sequence from 1 to 10 (see Figure 3). At each stop of the sweeping line (e.g., 1, 2,. . .), the algorithm computes the mean value in the $\mathbf{y}$-axis for the two trajectories (called *drivers*) and generates a "location update" (x, y_{avg}, t) for the dummy trajectory. At the end, the algorithm applies the same reconstruction strategy that was used for the real trajectories of the dataset, to decide how consecutive location updates will be connected to each other. Once created, trajectory $\mathbf{f}$ is added both to the database (for coherence purposes) and to the current answer set. As a final comment, notice that the generated dummy trajectories can be chosen at following iterations of the trajectory interpolation algorithm as drivers for the construction of new dummies.

3.5 Handling of attacks

In what follows, we motivate the two types of attack that a malevolent user has at his or her disposal to compromise the dataset, and we provide a methodology that aims at shielding the proposed query engine against such attacks.

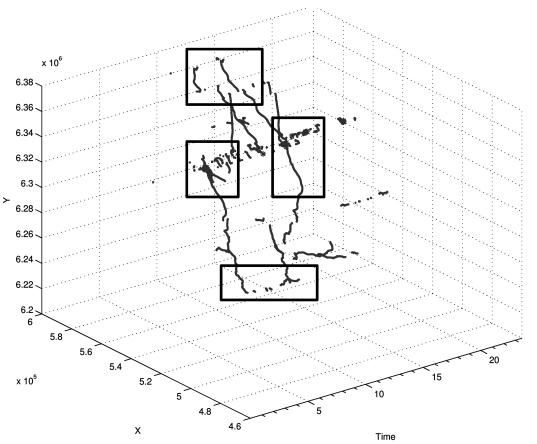


Figure 5: Spatiotemporal auditing to handle user identification.

3.5.1 User identification attack

In the user identification attack, the snooper creates a set of queries that overlap in the set of attribute-value pairs in an attempt to increase his/her confidence regarding the real identity of a user. A set of approaches for the handling of user identification in statistical databases in presented in [3]. In this paper we are interested in the formulation of this attack by means of queries that expose trajectory data. Consider the range query of Figure 2(i) that returns the trajectories of region `A`. By assuming 4-anonymity, the query engine produces a dummy trajectory $\mathbf{p}$ that is the result of interpolation on trajectories $\mathbf{o}$ and $\mathbf{q}$. The real trajectories $\mathbf{o}, \mathbf{q}$ and $\mathbf{r}$ along with the dummy trajectory $\mathbf{p}$ satisfy 4-anonymity. Now, suppose that a snooper wants to identify whether trajectory $\mathbf{o}$ is real or dummy. To do so, he or she selects a region `B` that is part of the previous region `A` and contains trajectory $\mathbf{o}$. Then, the snooper poses a query to the engine, requesting the available trajectories in region `B`. If the query engine answers this query, it will return trajectory $\mathbf{o}$ and part of trajectory $\mathbf{p}$, along with two dummy trajectories for the offering of 4-anonymity in region `B`. As a result, and since the inquirer knows the trajectories of region `A`, his or her confidence regarding trajectory $\mathbf{o}$ will dramatically increase from $1/4$ (as of region `A`) to $1/2$ (due to region `B`). It is evident that such an attack can effectively breach the anonymity model that is enforced by the query engine.

To prohibit the user identification attack, we take advantage of the auditing mechanism [3; 4], as applied to relational data, and transform it to operate on spatiotemporal data. In auditing, the system holds up-to-date logs of all the queries initiated by each user and checks for a compromise, whenever a new query is issued by the user. In our case, we need to keep track of the spatiotemporal regions containing the trajectories that formulate the answer to each query. Then, whenever the user poses a new query, the query engine scans the user history to examine if the spatiotemporal area in the new query overlaps with the area of a previous query. In this case, the system denies answering the query. Otherwise, the system proceeds to answer the query and subsequently stores the corresponding spatiotemporal area in the user history. The area to be stored is the Minimum Bounding Rectange (MBR) of the trajectories that appear in the answer set (including the dummies). Furthermore, all computed MBRs are appropriately indexed to allow for efficient retrieval. For a set of possible indexing schemes (e.g.,

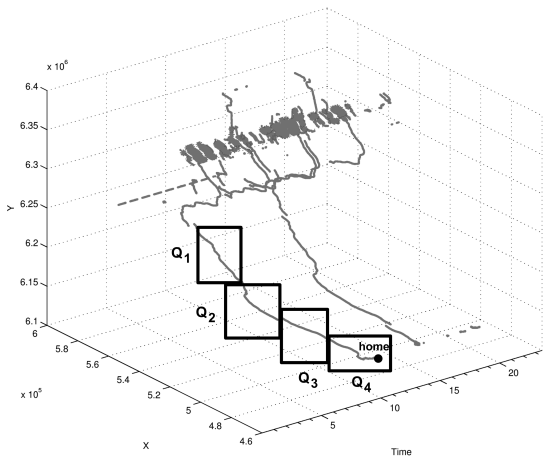


Figure 6: Spatiotemporal auditing to handle sequential tracking.

STR-tree and TB-tree), see [6]. An example of the operation of this approach appears in Figure 5, where four MBRs capture the areas of denial for subsequent queries issued by a specific user.

An extension of the auditing mechanism is applied on trajectories that belong to easily identifiable people, such as Bill Gates or Lucky Luck! Specifically, to avoid providing trajectory information for individuals who can become targets of attack, our approach denies answering queries in areas that contain VIP's trajectories.

3.5.2 Sequential tracking attack

In the sequential tracking attack, the snooper attempts to "follow" a user trajectory in the database, through a set of focused queries on spatiotemporal regions that are adjacent[4] to each other. If no special action is taken on behalf of the query engine, the attacker will be able to (i) increase his or her confidence regarding real/dummy trajectories provided as answers of past queries, and (ii) be certain that the trajectory that he or she follows belongs to a real user and is not dummy.

Consider the range queries Q1-Q4 of Figure 6. When a snooper poses query Q1 to the database, the query engine will return the trajectory shown, along with $K-1$ other trajectories to satisfy K-anonymity. Then, the snooper poses query Q2, requesting the trajectories that exist in a region that is adjacent (here touches) to that of Q1. The real trajectory that was part of the answer set of Q1 will also appear in Q2 and the depicted movement will continue smoothly, as is shown on Figure 6. To satisfy K-anonymity in the region of Q2, the query engine will retrieve any other real trajectories in this region and (possibly) generate some dummies. However a naïve dummy generation algorithm, will produce dummies by taking into consideration only the region of Q2 and not that of Q1. As a result, the snooper will easily spot the dummies among the real trajectories and will also be certain that the trajectory that he or she "follows" is real; no dummy trajectory spans from the region of Q1 to that of Q2. The same holds throughout the whole route from Q1 to Q4 and, as a result, the snooper succeeds in tracking a real user to his/her destination (indicated in Q4).

The situation we just described is captured in Figure 7(i), where 3-anonymity is considered. The trajectories appearing with solid lines are real, while those with dotted lines are dummies. As one can observe, the answer of Q2 uncovers the dummy trajectory in Q1, since the "user" movement in Q1 abruptly discontinues in Q2. As a result, the confidence of a snooper regarding the real trajectories in Q1 increases from $1/3$ to $1/2$. Furthermore, if the user has knowledge of the dummy generation scheme then he/she is confident that the sole trajectory that spans from the region of Q1 to that of Q2 is real. Thus, by continuing the sequential tracking attack the snooper achieves to track down a real user appearing in the dataset.

[4]By adjacency we mean that the MBRs of the two regions either touch or are within a short distance from each other.

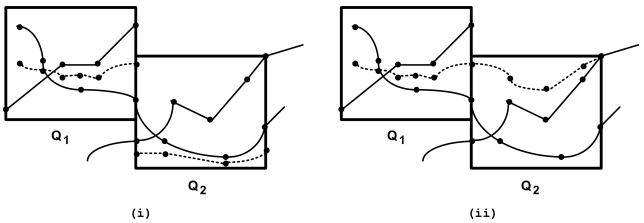


Figure 7: Defending the database from sequential tracking.

To prohibit sequential tracking, we use auditing (as before) to identify queries to regions that lie in the spatiotemporal neighborhood of past queries. If such a query is posed, we follow the approach shown in Figure 7(ii). Specifically, our proposed algorithm aims at confusing the snooper regarding the real and the dummy trajectories by continuing the movement of dummies in adjacent regions. Although alternative solutions may be devised to tackle this attack, we consider the proposed approach to be both simple and effective. First, the confidence of the snooper regarding the real trajectories in Q1 remains the same, since the existence of a dummy trajectory cannot be inferred. Second, due to the dummy generation algorithm that designates the behavior of the dummies, the snooper cannot be confident that the trajectory he or she follows belongs to a real user and is not dummy. Thus, given an adequate value of K in K-anonymity, the snooper will be discouraged to perform sequential tracking since the odds of "following" a real user will be considerably low.

4. ALGORITHMS

The methodology presented in Section 3 is implemented as a set of algorithms, utilized by the query engine to undertake the necessary tasks for privacy preservation. In what follows, we explain the operation of these algorithms and shed light on their key steps.

Listing 1 collects algorithms that operate on queries for non trajectory data. Such queries are handled differently based on whether they are count queries or queries for aggregate statistics. In the first case, the query engine directly services the user query. In the second case, it augments the SELECT statement with a counter of the returned records, prior to servicing the query. Furthermore, in aggregate queries, *func(attr)* signifies the use of an SQL aggregate function, such as `AVG`, `SUM`, `MAX` or `MIN`, applied on attribute *attr* of the data table. In both cases, the WHERE clause in the SELECT statement copies the condition *cond* from the user query. After servicing the query, the query engine computes the number of records from the database that were involved in the answer. It then proceeds to disclose the answer to the user, only if it satisfies the requirements of K-anonymity.

Listing 2 contains algorithms that handle range and nearest neighbors queries. From the algorithmic standpoint, we make a slight distinction between distance queries (e.g., **Q4**) and range queries (e.g., **Q2**). Specifically, distance queries are transformed into range equivalents by constructing the range of interest as a circle centered

Listing 1 Computation of count/aggregate non-trajectory queries.

```
1: function COUNT-QUERY(cond, K)
2:     conn ← getConnection(string)                          ▷ Connect to DB
3:     stmt ← conn.createStatement()
4:     rset ← stmt.executeQuery("SELECT count(*)
                                        FROM data WHERE cond)
5:     ans ← rset.getValue(1)
6:     if ans ≥ K then return ans
7:     else
8:         raise_exception("K-anonymity violation")
```

```
1: function AGGREGATE-QUERY(func, attr, cond, K)
2:     conn ← getConnection(string)                          ▷ Connect to DB
3:     stmt ← conn.createStatement()
4:     rset ← stmt.executeQuery("SELECT count(*), func(attr)
                                        FROM data WHERE cond")
5:     ans ← rset.getValue(1)
6:     if ans ≥ K then return rset.getValue(2)
7:     else
8:         raise_exception("K-anonymity violation")
```

Listing 2 Computation of range/distance/k–NN trajectory queries.

```
1: function RANGE-QUERY(cond, Sgeo, t_start, t_end, K)
2:     if IN_HIST(Sgeo, t_start, t_end) then
3:         raise_exception ("Privacy threat")
4:     conn ← getConnection(string)                          ▷ Connect to DB
5:     cstmt ← conn.prepareCall(RANGE_QR(Sgeo, t_start, t_end))
6:     cstmt.execute()                                       ▷ Execute PL/SQL routine
7:     if #recs(T) ≥ K then
8:         return T.view(cond)
9:     FAKE-GEN(K −#recs(T), Sgeo)
10:    UPDATE_HIST(Sgeo, t_start, t_end)
11:    return T.view(cond)
```

```
1: function kNN-QUERY(cond,P,t_start, t_end, k, K)
2:     create temporary table tmp like data
3:     tmp.movement ← user trajectories in [t_start, t_end]
4:     INSERT INTO T (SELECT * FROM tmp WHERE
                          NN(tmp.f_trajectory(), P) = 'true' AND rownum ≤ k)
5:     if IN_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end) then
6:         raise_exception ("Privacy threat")
7:     if #recs(T) ≥ K then
8:         return T.view(cond)
9:     FAKE-GEN(K −#recs(T), mbr(aggr(T.f_trajectory())))
10:    UPDATE_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end)
11:    return T.view(cond)
```

```
1: function DISTANCE-QUERY(cond, X_p, Y_p, d, t_start, t_end, K)
2:     Sgeo ← create_circle (X_p, Y_p, d)
3:     RANGE-QUERY(cond, Sgeo, t_start, t_end, K)
```

```
1: PL/SQL procedure RANGE_QR(Sgeo, t_start, t_end)
2:     DECLARE
3:     CURSOR mobjs IS SELECT movement FROM data
4:     mo, mlim    Moving_Object
5:     geom, g     Geometry
6:     BEGIN
7:     open (mobjs)
8:     loop
9:         fetch mobjs INTO mo
10:        exit when mobjs%NOTFOUND
11:        mlim ← mo.at_period(t_start, t_end)
12:        geom ← mlim.f_trajectory()
13:        g ← intersect(geom, Sgeo)
14:        if g ≠ null then
15:            INSERT INTO T VALUES (..., intersect(mo, g))
16:    close(mobjs)
17:    END
```

Listing 3 Computation of landmark(s)/route trajectory queries.

```
1: function LANDMARK-QUERY(cond, t_start, t_end, K, geom =
   < P_1, P_2, ..., P_r >)
2:     create temporary table tmp like data
3:     tmp.movement ← user trajectories in [t_start, t_end]
4:     INSERT INTO T (SELECT * FROM tmp WHERE
                          Contains(tmp.f_trajectory(), geom.P_1) = 'true' AND
                          Contains(tmp.f_trajectory(), geom.P_2) = 'true' AND
                      ...AND Contains(tmp.f_trajectory(), geom.P_r) = 'true')
5:     if IN_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end) then
6:         raise_exception ("Privacy threat")
7:     if #recs(T) ≥ K then
8:         return T.view(cond)
9:     FAKE-GEN(K −#recs(T), mbr(aggr(T.f_trajectory())))
10:    UPDATE_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end)
11:    return T.view(cond)
```

```
1: function ROUTE-QUERY(cond, t_start, t_end, K, R)
2:     create temporary table tmp like data
3:     tmp.movement ← user trajectories in [t_start, t_end]
4:     INSERT INTO T (SELECT * FROM tmp WHERE
                          Contains(tmp.f_trajectory(), R) = 'true')
5:     if IN_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end) then
6:         raise_exception ("Privacy threat")
7:     if #recs(T) ≥ K then
8:         return T.view(cond)
9:     FAKE-GEN(K −#recs(T), mbr(aggr(T.f_trajectory())))
10:    UPDATE_HIST(mbr(aggr(T.f_trajectory())), t_start, t_end)
11:    return T.view(cond)
```

Listing 4 Search/Update the user history of queries.

```
1: function IN-HIST(Sgeo, t_start, t_end)
2:     conn ← getConnection(string)                          ▷ Connect to DB
3:     stmt ← conn.createStatement()
4:     rset ← stmt.executeQuery("SELECT count(*) FROM
       hist WHERE (t_end ≥ t_A ∧ t_start ≤ t_B) ∧ (Overlaps(geom, Sgeo) = 'true'
                                        ∨ Neighbors(geom, Sgeo) = 'true' ")
5:     if rset.getValue(1) = 0 then return true
6:     return false
```

```
1: Procedure UPDATE-HIST(Sgeo, t_start, t_end)
2:     conn ← getConnection(string)                          ▷ Connect to DB
3:     stmt ← conn.createStatement()
4:     rset ← stmt.executeUpdate("INSERT INTO hist VALUES
                                        (Sgeo, t_start, t_end) ")
```

Table `hist`:

geom	t_A	t_B

at the provided location P and having a radius of d (requested distance). To service a range query, the algorithm first ensures that it involves a spatial/temporal or spatiotemporal region that blocks both the user identification and the sequential tracking attacks. If this so happens, then with the aid of a PL/SQL procedure, the algorithm captures the movement of all users in the selected region and stores it in a table T (similar to the one that holds the data). Table T collects all user records having trajectories in the selected region, along with the user (sub)-trajectories in this region. Then, the algorithm counts the number of records in table T to identify if they suffice for the requirements of K-anonymity. If this is the case, then the algorithm returns the user requested view of the results (line 8). Function T.view(*cond*) corresponds to a select query in T using condition *cond* in the WHERE-clause. On the other hand, if the retrieved records in table T are insufficient for K-anonymity, the algorithm requests the generation of the needed fakes and subsequently updates the history of user queries to include the answer region of the serviced query. This update is necessary to block any future attempts of the user to compromise the dataset.

Queries requesting the k nearest trajectories to a given point P, are handled similarly to the range queries. The k-NN–Query function, presented in Listing 2, uses a temporary table tmp to store (apart from all non-trajectory attributes), all user trajectories in the specified time interval (lines 2-3). By using the information in this table, the algorithm identifies the k nearest trajectories to point P and stores the corresponding records to table T. The user history of queries is then examined to identify a potential match in the spatiotemporal region, where the spatial dimension is the MBR of the (sub)-trajectories of all k nearest neighbors of P. If the query is found to be "safe" with respect to the two types of attack, the algorithm checks if it fulfils the requirements of K-anonymity. If it does, then the query is answered and the user history of queries is updated. On the other hand, if less than K records exist in table T, the algorithm generates the necessary fakes in the computed spatiotemporal region.

Listing 3 collects the necessary algorithms for the handling of landmark and route queries. The followed approach is similar to the k-NN–Query algorithm. First, a temporary table tmp stores all records in the database, involving user (sub)-trajectories in the specified time period. Then, a table T, similar to tmp, is populated with all the records from tmp that contain either trajectories that pass through a number of pre-specified points P (landmark query), or trajectories that follow a pre-specified route *R* (route query). Finally, table T is augmented (if necessary) with fake records, the user history of queries is updated and the K-anonymous query answer is returned to the user.

Listing 4 presents algorithms that operate on the user history of queries. The query history of a user is captured in a table hist that contains a spatial attribute *geom*, storing the spatial region of the user query, and two timestamp variables $\mathbf{t_A}$ and $\mathbf{t_B}$, storing the temporal interval of the query. An update operation to the user history of queries is equivalent to the insertion of a record to table hist, indicating the spatiotemporal region of the query. On the other hand, the check for a potential attack is performed by identifying the existence of any record in hist that involves a spatiotemporal region that either overlaps with the spatiotemporal area of the user query, or it neighbors with it. Both cases indicate a potential privacy violation and result in the denial of the query engine to answer the query.

Listing 5 presents the algorithm that is used for the generation of the fake records. First, the algorithm generates two temporary dummies, $\mathbf{u}$ and $\mathbf{v}$ (see Figure 3), to make up for the uppermost and the lowermost trajectories in the provided spatial region *Sgeo* (lines 2-4). The dummy trajectories are subsequently included into empty records T_u and T_v and inserted into table T (lines 5-6). Then, the algorithm generates the required number of fake records and sets the values in all their numerical attributes (lines 7-19). Specifically, for each numerical attribute, the algorithm uses the real records in T to compute its mean and deviations. Following that, the algorithm performs a random partitioning of the fake records into pairs and for each pair it identifies a value near the attribute mean and uses this value to perform the attribute-value assignment. After assigning values to all the numerical attributes in the fake records, the algorithm proceeds to the generation of trajectory dummies (lines 20-31). To accomplish that, for each fake record it randomly selects two records from table T and orders the corresponding trajectories in an ascending order of their location updates (lines 21-22) with respect to the **x**-coordinate. Then, for each location update, the algorithm identifies the corresponding point in the other trajectory that has the same **x**-coordinate and computes a "dummy" location update as the mean of the two real ones (with respect to the **y**-coordinate). This "dummy" location update is kept in a list dummy−lu (line 28). After collecting all the "dummy" location updates, the algorithm proceeds to reconstruct the movement of the dummy by selecting consecutive location updates from dummy−lu and applying a trajectory reconstruction function *rfuc* (e.g., like the ones of Figure 4). As a final step, the reconstructed dummy trajectory is assigned to the corresponding fake record and included in table T to be a candidate for the next iteration of the algorithm. The same operation holds for the generation of all the necessary dummies, their assignment to fake records and subsequent inclusion in table T. After producing all the fake records, the algorithm removes from table T the temporary fakes T_u and T_v.

Listing 5 Generation of M fake records in spatial region *Sgeo*.

1: **procedure** FAKE-GEN(M, *Sgeo*)
2: compute y_{min}, y_{max} in *Sgeo*
3: compute t_{min}, t_{max} in T.movement
4: generate temporary dummies $\mathbf{u}(t)= y_{max}$, $\mathbf{v}(t)= y_{min}$, where $t \in [t_{min}, t_{max}]$ (**u**,**v** have constant movement)
5: generate empty records T_u, T_v and associate with **u**,**v**
6: insert T_u, T_v into table T
7: generate M empty fake records $f_1 \ldots f_M$
8: **if** M%2 = 1 **then**
9: select f_1
10: **for each** numerical attribute $\alpha \in A$ **do**
11: $f_{1.\alpha} \leftarrow avg(\alpha$,T) ▷ avg over the real records
12: **for each** numerical attribute $\alpha \in A$ **do**
13: $d_{min} \leftarrow avg(\alpha, \text{T}) - min(\alpha, \text{T})$
14: $d_{max} \leftarrow max(\alpha, \text{T}) - avg(\alpha, \text{T})$
15: randomly group fake records into pairs $p_1 \ldots p_{M/2}$ ▷ exclude f_1 if M is odd
16: **for each** pair $p_i(f_j, f_k) \in [p_1 \ldots p_{M/2}]$ **do**
17: $d \leftarrow$ random value in $[0, min(d_{min}, d_{max})]$
18: $f_{j.\alpha} \leftarrow avg(\alpha, \text{T}) - d$
19: $f_{k.\alpha} \leftarrow avg(\alpha, \text{T}) + d$
20: **for each** fake record $f_j \in [f_1 \ldots f_M]$ **do**
21: randomly select two records T_p, T_q from table T
22: **list** $L \leftarrow$ sort$_{\mathbf{x}-coord}$ (location-updates($T_p \cup T_q$))
23: **for each** location update $lu \in L$ **do**
24: **if** $lu = (x_p, y_p, t_p) \in T_p$ **then**
25: $lu' \leftarrow (x_p, T_q.\text{movement}(x_p)/2, t_p)$
26: **else if** $lu = (x_q, y_q, t_q) \in T_q$ **then**
27: $lu' \leftarrow (x_q, T_p.\text{movement}(x_q)/2, t_q)$
28: insert (**list** dummy−lu, lu')
29: **for all** consecutive $lu_1, lu_2 \in$ dummy−lu **do**
30: update (f_j.movement, MRECON(*rfunc*, lu_1, lu_2))
31: insert fake record f_j to T
32: remove T_u, T_v from table T

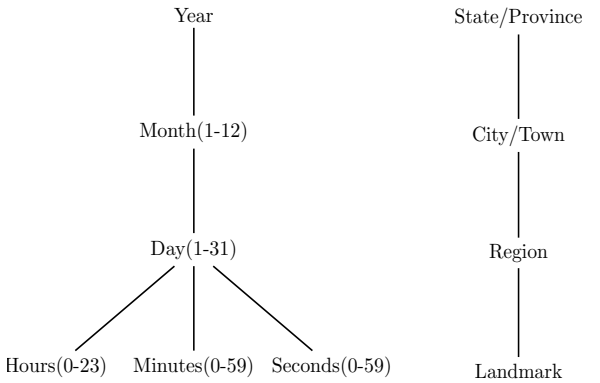


Figure 8: Query specification through time/space hierarchies.

5. QUERY EXTENSIONS

The query formulation mechanism (presented in Section 3.2), suffices for laying down queries involving specific landmarks (e.g., **Q1** or **Q6**), or spatial areas of a common granularity (e.g., **Q4**, **Q5**). Furthermore, it effectively handles queries involving temporal data in the form of specific times or time intervals during a given calendar day (e.g., **Q2**). Both these features, when combined together, allow the specification of spatiotemporal queries at a fine granularity (e.g., **Q3**, **Q7** or **Q9**). However, there are many occasions when the inquirer wants to retrieve knowledge at different levels of granularity, possibly coarser than the one of the previous queries. For example, a user might want to pose queries like:

Q10 "Find all user trajectories this Monday".

Q11 "Find all user trajectories in Minneapolis, MN".

Q12 "Find all users that moved on January 2008 in the city center of Minneapolis, MN".

Answers to those queries allow for an advanced statistical analysis of the available movement data and can be proved valuable to draw conclusions regarding the periodicity of certain events. To support the specification of such queries, we introduce two concept hierarchies that depict space and time at different levels of granularity (see Figure 8). By using the available levels of granularity, the query engine can effectively handle queries that skip one or more levels of the spatial/temporal hierarchy by first computing the needed aggregations and performing the required materializations in the database. For example, to answer query **Q12** the query engine will first retrieve the region that corresponds to the city center of Minneapolis by performing an aggregation over all appropriate streets and landmarks that belong to the city center. Then, it will acquire all trajectories that pass through the computed region. Finally, it will remove from the answer set all the trajectories with timestamps that indicate a month other than January and a year other than 2008. Of course, the necessary computations can take place in a different sequence and still lead to the same result. As one can observe, the organization and the distribution of the trajectory data are the major criteria to identify which, among the possible sequences of computations, is bound to perform best.

6. QUALITATIVE ANALYSIS

In this section, we provide a qualitative analysis of our proposed approach along three principal directions; database distortion, blocking of attacks and computational complexity.

6.1 Database distortion/blocking of attacks

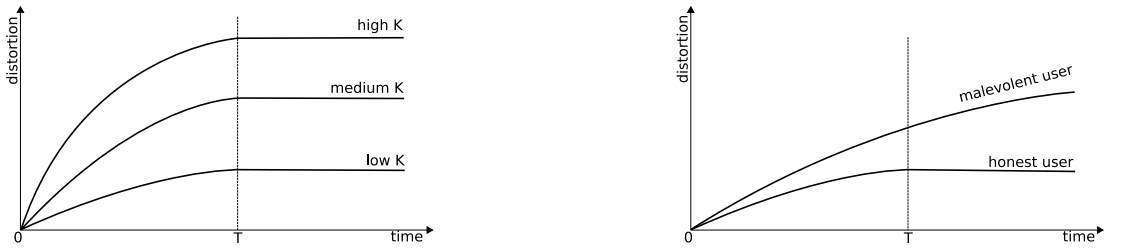
Figure 9 presents the distortion of the database caused by the incorporation of fakes. The distortion is measured as the percentage of fake records in the database at any given time. Figure 9(i) plots the distortion of the database with respect to the posed user queries (in time), for different requirements of anonymity. As expected, lower values of K result in a smaller distortion of the database, since less fakes are needed to meet the requirements of K-anonymity. Furthermore, for any value of K there exists some time instance T at which the fakes in the dataset suffice to answer a reasonable amount of user queries, without the need of generating additional fakes. From this point on, the database is typically distorted at a much lower rate than previously. Figure 9(ii) plots the distortion that is inflicted in the database by two users who are assigned the same K in anonymity. As one can observe, a significant difference in the distortion of the database that is caused by a user (with respect to the rest of the population) may suggest the user's attempt to compromise the database. This is due to the fact that snoopers try to spot subjects and thus pose a large series of focused queries, leading to more fakes being generated by the query engine.

Figure 10 presents two mechanisms to distinguish honest from malevolent users. Specifically, Figure 10(i) is based on the premises that snoopers initiate sequences of trajectory queries involving small spatiotemporal regions that are close to each other. For each user of the system we generate a plot capturing the spatiotemporal regions in the user history of queries, ordered by proximity to each other (**x**-axis) versus the sequential numbers that indicate the order of queries in the user history (**y**-axis). Then, points that lie close to each other in the plane correspond to sequential user queries involving neighboring regions. Such a behavior is typical for snoopers, while atypical for honest users. Another good indicator of malevolent behavior, is the existence of numerous user queries in sparsely populated regions (Figure 10(ii)). By using graphs such as the ones presented above, the administrators of the system are capable of blocking most attempts of snoopers to compromise the database.

6.2 Computational complexity

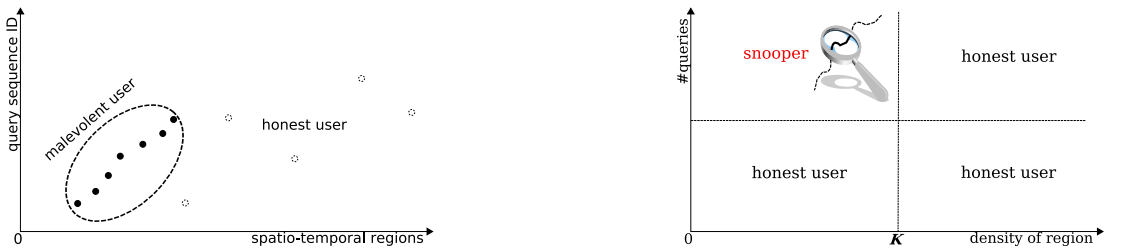
The queries, supported by the query engine, experience different requirements at terms of computational cost. Count and aggregate queries require $\Theta(d)$ runtime, where d is the number of records in the *data* table. Range and distance queries operate similarly. They are both based on the RANGE_QR procedure that relies on an R–tree, experimentally proven to have polylogarithmic behavior. On average, RANGE_QR requires $O(d\ (polylog\ d) + o)$, where *polylog* is applied on the number of trajectories on the *data* table and o is the number of elements returned by the R–tree[5]. In k–NN queries the selection of the user trajectories at the requested time period requires $O(log\ d + o)$ (worst case), given a B–tree index on the timestamp data. The actual identification of the nearest neighbors requires on average $O(s+log\ d)$, where s is the number of trajectories in this time period. The same is the cost for the trajectories selection process in landmarks/route queries. All the aforementioned types of queries, with the exception of the count/aggregate queries, have an additional cost of searching and updating the user history. A search in the user history of queries incurs $O(polylog\ h + o)$ cost, where h is the size of the user history and o is the number of elements returned by the R–tree index. On the other hand, the cost of updating the user history is in the worst case $O(log\ h)$ due to the insertion of the corresponding region to the R–tree. Finally, the cost of fake records generation equals the cost of creating M empty

[5]Searching in R-trees/B–trees has an additional multiplicative constant cost that depends on the page size of the underlying index.



(i) distortion with respect to the value of K. (ii) distortion with respect to the user queries.

Figure 9: Database distortion due to the incorporation of fake records.



(i) detection of the user tracking attack. (ii) identification of suspicious user behavior.

Figure 10: Identifying potential attacks based on the history of user queries.

records, that is $O(M)$, updating their values for the A numerical attributes, that is $O(A\ (T+M))$ (T is the number of trajectories in the *Sgeo* region), and producing the M dummy trajectories, that is $O(M\ L\ log\ L + M\ f\ L + M\ log\ T)$, where L is the number of location updates in $T_p \cup T_q$ and f is the cost of deciding whether a point lies in a given trajectory. As a final comment, it is essential to mention that the presented analysis is based on empirical estimates of the actual runtime costs, since we have no access to the implementation details of Oracle.

7. CONCLUSIONS

In this paper, we presented a privacy-aware trajectory tracking query engine that offers K-anonymous answers to user queries. Our proposed engine allows the data to stay in house and supports a large variety of query types, involving both trajectory and non-trajectory data. To offer K-anonymity in trajectory queries, the engine relies on the generation of fake records involving dummy trajectories. The query engine offers strict guarantees about what can be found by untrusted third parties and is capable of blocking two types of potential privacy breaches, namely *user identification* and *sequential tracking*. Qualitative analysis of our approach shows its effectiveness towards preventing malevolent users from compromising the dataset, while causing small distortion.

8. REFERENCES

[1] O. Abul, M. Atzori, F. Bonchi, and F. Giannotti. Hiding sensitive trajectory patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 693–698, 2007.

[2] O. Abul, F. Bonchi, and M. Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, 2008.

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

[4] L. J. Hoffman. *Modern Methods for Computer Security and Privacy*. Prentice-Hall, Englewood Cliffs, N. J., 1977.

[5] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. In *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 194–205, 2005.

[6] Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras. *Advanced Database Indexing*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. (Chapter 7).

[7] N. Pelekis and Y. Theodoridis. Boosting location-based services with a moving object database engine. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access (MobiDE)*, pages 3–10, 2006.

[8] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppresion. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 384–393, 1998.

[9] M. Terrovitis and N. Mamoulis. Privacy preservation in the publication of trajectories. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM)*, 2008.

[10] B. Yu and S. H. Kim. Interpolating and using most likely trajectories in moving-objects databases. In *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA)*, pages 718–727, 2006.