UNIVERSITY OF PIRAEUS

School of Information and Communication Technologies

Department of Informatics

Thesis

Thesis Title	Privacy-aware urban mobility data harvesting Συλλογή αστικών δεδομένων κίνησης διασφαλίζοντας την ιδιωτικότητα των αντικειμένων
Student's name-surname	Ioannis Athanasopoulos
Father's name	Dimitrios
Student's ID No.	Π19005
Supervisor	Yannis Theodoridis, Professor

Ημερομηνία Παράδοσης **Ιούλιος 2024**

Copyright

Acknowledgements

In this section, I would like to express my deepest gratitude to those who contributed to the completion of this thesis.

First and foremost, I would like to thank my supervisor, professor Yiannis Theodoridis, for the guidance and continuous support throughout the research process, as his knowledge on the field was of great importance for the success of this project.

Additionally, I would like to express my gratitude to fellow researchers for their ideas and perspective, as they helped me lay the foundations and generally shape this project.

Finally, I would like to thank my family and friends for their unending support throughout this whole journey.

Περίληψη

Με την διαρκή άνοδο των έξυπνων πόλεων και την ευρεία χρήση των υπηρεσιών τοποθεσίας από κινητές συσκευές, αυξάνεται ο προβληματισμός σχετικά με τον κίνδυνο παραβίασης της ιδιωτικότητας από πληροφορίες που μπορούν να εξαχθούν από τις ανθρώπινες τροχιές. Η συγκεκριμένη εργασία με τίτλο «Συλλογή αστικών δεδομένων κίνησης διασφαλίζοντας την ιδιωτικότητα των αντικειμένων» επικεντρώνεται στον τρόπο που η πληροφορία μπορεί να εξαχθεί από τις επιθέσεις ιδιωτικότητας, και συνεπώς ερευνά τον πιθανό κίνδυνο που εγκυμονεί η συλλογή δεδομένων κίνησης σε αστικά περιβάλλοντα. Αρχικά η μελέτη αναγνωρίζει και αναλύει τους συχνότερους τύπους επιθέσεων που μπορούν να εκτελεστούν σε πραγματικές καταστάσεις και τους τύπους πληροφορίας που μπορούν να χρησιμοποιηθούν για την ταυτοποίηση των ατόμων. Στην συνέχεια προτείνεται η υλοποίηση στην προγραμματιστική γλώσσα Rust της διαδικασίας του υπολογισμού του κινδύνου των επιθέσεων ιδιωτικότητας. Τέλος, πέρα από κάποια πειράματα για την αξιολόγηση της επίδοσης της προτεινόμενης υλοποίησης τόσο ως προς τον χρόνο όσο και ως προς την ακρίβεια, η εργασία δείχνει την σύγκρισή της με την πλέον σύγχρονη αντίστοιχη υλοποίηση, το Scikit-mobility.

Abstract

With the growth of smart cities and broadening usage of location-based services with mobile devices, there is rising concern about privacy risks of information that can be inferred from human trajectories. The present thesis entitled "Privacy-aware urban mobility data harvesting" focuses on how sensitive information can be inferred via various privacy attacks; hence, it presents research into potential privacy risks of urban mobility data harvesting. The study first identifies and analyzes common types of privacy attacks that can be executed in real-world scenarios and the types of information that can be accessed and used to identify individuals. This paper subsequently implements, in the Rust programming language, the procedure for evaluation of such risks of privacy attacks. Finally, besides some experiments to assess the suggested implementation's performance in terms of time and accuracy, this thesis shows the comparison of this implementation with the existing state-of-the-art implementation, Scikit-mobility.

Contents

List of Figures

List of Tables

Introduction

Background and Context

With the rise of smart cities, connectivity enters a whole new dimension whereby urban environments depend more and more on data-driven technologies that make services more efficient, sustainable, and of a higher quality for citizens. At the heart of this digital transition lies collecting and processing data on urban mobility that offers much more valuable insight and reaction to city planners, businesses, and service providers in relation to the movement of individual people in a city. This can usually be compiled through location-based services provided within mobile devices, offering unparalleled insights for traffic management, public transportation optimization, and urban planning.

Problem Statement

However, the ever-increasing quantity of urban mobility data raises serious privacy concerns. Sensitive information, such as home and work location, daily routines, mobility patterns and social interactions, can be disclosed quite easily from data gathered in human trajectories. Even when anonymized, mobility data is often vulnerable to privacy attacks wherein malicious actors can re-identify individuals by correlating trajectories with publicly available information. The latter has increasingly brought attention to the risks of harvesting urban mobility data, particularly in regard to the protection of personal privacy.

Objectives

The present thesis, entitled "Privacy-aware Urban Mobility Data Harvesting", tries to address these concerns by researching how sensitive information can be inferred through privacy attacks on mobility data. This research has identified through case study analysis various possible kinds of privacy attacks and the mechanism of how such attacks can make use of vulnerability from the mobility dataset. The thesis also shows how these attacks can be implemented using the Rust programming language to evaluate the risk each of them poses.

In particular, the implementation developed in this work is benchmarked against a state-of-the-art privacy attack framework called Scikit-mobility that has seen extensive usage in mobility data analysis. This thesis verifies, through a series of experiments, how time-efficient and accurate the proposed solution is and provides insight into how well it performs compared to existing methodologies. This dissertation aims to point out the soft spots of location-based services at the intersection of data on urban mobility and concerns about privacy, and to provide possible mitigation strategies that improve privacy in 'smart cities.' From this will come meaningful implications for policymakers, city planners, and developers currently working toward the creation of more secure and privacy-sensitive urban environments.

Literature Review

Mobility Analytics

During the past years, mobility analytics has gained a significant interest since almost every domain bases its decisions on data. Spatial and spatio-temporal data represent one of the grounds for determining mobility patterns; hence, the importance of its analysis and pre-processing. Interest in performing spatial and spatio-temporal analysis of mobility data is considered an opportunity to create new structures of data and management systems that properly operate such complex datasets. Some examples include MobilityDB, which is a database that deals with data generated by moving objects and Dragoon, which is an online and offline analytics trajectory management system. These systems allow for the creation of efficient data organization, aggregation, and analysis for mobility data.

Moreover, integrated tools are needed that support the whole process of data analysis. For example, TransBigData is a Python package for processing, analysis, and visualization of spatio-temporal big data. The development of such tools illustrates the trend toward automation and simplification of mobility data analysis in a world where real-time and large-scale processing becomes more central.

Preprocessing

The major steps in analyzing mobility data involve the preprocessing step. Preprocessing assures that data is clean and accurate; hence, the quality of analysis and results improves considerably. The detection and removal of outliers and missing values make a dataset more reliable in this stage. Such toolkits as CloudTP, a cloud-based trajectory data preprocessing framework, and PTRAIL-a Python package that offers filtering and feature extraction among other preprocessing steps, are also common tools to clean up the mobility datasets.

CloudTP transforms raw GPS logs into organized trajectories through a series of steps: noise filtering, trajectory segmentation, map matching and index building. This cleans up the data, puts it into an ordered structure, and makes it ready for analysis. PTRAIL joins the few preprocessors that support parallel computations and the processing of trajectory data, hence highly scalable when dealing with large volumes of data. As one can easily understand, preprocessing is of utmost importance, as any successive mobility analysis is contingent upon the accuracy and reliability that this step guarantees.

Privacy

Privacy concerns are central to mobility data analysis. By their very nature, mobility data may reveal sensitive patterns about the moving habits of individuals and hence require the development of sound privacy-preserving techniques. With mobility data harvesting becoming increasingly widespread, privacy risks pertaining to data processing and identification of subjects will have to be carefully managed.

To mitigate these risks, various privacy-preserving techniques have been developed. Some typical methodologies include generalization and k-anonymity, in which the data is made more general to protect the individual identity, the execution of privacy attacks to study the vulnerability of datasets and the modelling of an adversarial behaviour as a mobility trajectory in order to discover the most impactful adversary path concerning the privacy risk posed to the individuals. These methods illustrate the variety of strategies available to balance the utility of mobility data with the need to protect individuals' privacy.

Scikit Mobility Privacy

The Scikit-mobility Privacy library probably represents the most well-known tool in the context of privacy-aware mobility data analysis. With more and more widespread exploitation of location-based services and the increasing diffusion of mobility data analysis across several heterogeneous settings, individual privacy requires careful safeguarding. Scikit-mobility Privacy allows one to evaluate the actual privacy risks by simulating the execution of various kinds of identification attacks on trajectory datasets, enabling one to perceive the actual vulnerability of individuals to privacy leaks.

This Python-based tool evaluates individual identification risk by conducting various attacks and at the same time studying data sensitivity. Although functional, Scikit-mobility Privacy has a margin of improvement regarding performance. The use of more performant languages such as Rust would make a huge difference in runtime and performance in general, since large datasets are usually in consideration.

Our Approach

Given (i) a dataset D consisting of the traces of a population of individual users (pedestrians, drivers, etc.), in the form of tuples *(oid, t, loc)*, where *oid* is the unique identifier of an individual, *t* is the timestamp of its trace, and *loc* is the respective location[1], (ii) an attack configuration *A*, and (iii) a minimum danger threshold *dt*, $0 \leq dt \leq 1$, the goal is to detect a set $D' \subset D$, consisting of those tuples in D that match A with respect to dt. By purpose, the above definition is generic enough to cover different attack configurations. Before listing them, we provide some necessary background definitions and subsequently we present the preprocessing procedure that the data undergo before the attacks are executed:

1. The frequency $freq_{loc}$ of a location loc and an individual with identifier oid in D is defined as follows:

The frequency $freq_{loc}$ of a location loc is the count of all the records that follow the format (oid, timestamp, location), where oid is the unique identifier of a single individual oid and location is the location loc.

$$freq_{loc} = \sum_{i \in D} f(i) \quad (\text{Eq. } 1)$$

$$where\ f(x) = \begin{cases} 1, & x.oid = oid \wedge x.loc = loc \\ 0, & else \end{cases}$$

2. The probability of identification *prob* for a tuple (oid, loc, freq) in *D* and a set *S* of individuals that are considered matches for that specific tuple is defined as follows:

$$prob = \frac{1}{|S|} \quad (\text{Eq. } 2)$$

3. The formula for the assignment of POIs to a set of (x, y) in *D*, with min_{lon} and min_{lat} being the minimum values of the longitude and latitude columns respectively, $step_{lon}$ and $step_{lat}$ being the fixed incremental step for neighboring grid cells for the longitude and latitude respectively and N being the dimension of the *NxN* grid used for clustering, is defined as follows:

$$POI = \frac{(x - min\_lon) * N^2}{step\_lon} + \frac{(y - min\_lat) * N}{step\_lat} \quad (\text{Eq. } 3)$$

Preprocessing

For the Preprocessing procedure, the purpose of it is to prepare the data for the privacy risk evaluation that they will undergo. This of course is a procedure consisting of multiple consecutive steps; however, it can be narrowed down to three major sub-procedures.

Trajectory Segmentation

In this step, the DataFrame D that was described earlier in the form of (oid, t, loc) is organized in unique individual trajectories and separate trips. Each trip is distinguished by the combination of *(oid, trip id)*, *oid* being the unique identifier of an individual and *trip id* being the unique identifier among the trips of the same individual. In this step, besides the addition of trip id, the DataFrame is enriched

[1] The granularity of the location of a trace is application-dependent and could range from finer, e.g. a pair (lat, lon) of geographical coordinates, to coarser, e.g. a region of interest (ROI) covering an area of several sq.m. The same holds for the type of location taken into consideration, since it could cover all locations visited by the individual or only the locations where he/she stopped.

with additional information regarding the speed, the bearing, whether the individual is stopped and the closest point of interest (POI) based on a grid-based clustering of the locations.

Origin and Destination Calculation

During this step, we enrich the DataFrame with two additional columns, regarding the origin and the destination POI of each point within a trip. All points of the same trip need to have the same origin and destination POIs, which will later be useful for the identification of the DataFrame's dangerous records. This step creates a final preprocessed version of the original dataset, containing every info needed for the final sanitization of the latter.

Knowledge Base Construction

The final sub-procedure is the creation of a unified knowledge base for all the attacks. This knowledge base is a DataFrame in the form of *(oid, poi id, freq), oid* being the unique identifier of an individual, *poi id* being the POI that has been visited by the aforementioned individual and *freq* being the respective number of visits for this POI.

Below, we can see a more detailed overview of the Preprocessing phase of this approach.

1. The first step is to find the minimum and maximum values of the longitude and latitude of the input DataFrame. These values will be the bounds of the grid that will be used for the grid-based clustering. Based on the bounds and the dimension of the grid, we also calculate the incremental step of neighboring cells, used in the POI assignment step.
2. An empty Trajectory Collection data structure is created for the separate trajectories to be stored in.
3. For each record of the input DataFrame, we perform the following steps:
    a. We firstly try to assign this record to an existing Trajectory object. If one does not exist, we create a new empty Trajectory object and start with the next record of the input. Else, extend the existing Trajectory object with a new point.
    b. If the timestamp of the record is the same as the Trajectory object's last element, we return the existing Trajectory and start with the next record, as there will be no significant change to be recorded.
    c. We calculate the speed and bearing from the Trajectory's last element, finding the distance covered and dividing by the time frame between the current and the previous record. If the speed exceeds a threshold for the maximum speed, we return the existing Trajectory object as is and continue with a new record, thus omitting the current record. If the speed is within bounds, we calculate if the speed is below the stoppage threshold and assign the value 1 if it is (and therefore is a stop point) or 0 if it is over the threshold (and thus considered moving).
    d. For the grid-based POI assignment procedure, there are four possible scenarios:
        i. If the point is a stop point and the last point was also a stopped one, we assign the previous POI as the value of the current one too.
        ii. If it is a stop point and the last one was a moving point, we calculate the nearest point with the formula presented in Eq. 3.
        iii. If it is a moving point and the last one was a stopped one, we assign the previous poi as the value of the current one too. This is of course logical, as it's the first moving point after the end of a trip and thus the individual will start from the same POI.
        iv. In any other case, more specifically a moving point after a moving point, we assign the number -2, as we don't need a number for this case.
    e. In this step, we calculate the trip ID of each record. If the previous point was a stop point and this is a moving one, we assume that this is a new trip with a trip ID of the

previous number of trips incremented by one, else the trip ID is the same as the previous record's trip ID.

f. We finally insert all this information to a new or existing Trajectory object and return it.

4. If the Trajectory returned already exists inside the Trajectory Collection data structure, we replace it with the new object created, else we append it inside the Trajectory Collection data structure.
5. In this step, we calculate the origin and destination of each trip. First of all, we create a DataFrame from the Trajectory Collection data structure and we group it by the attributes "oid" and "trip". Subsequently, we do the following for each group:
    a. We find the first and last POI visited of the POI column of each group.
    b. We create a two-column DataFrame containing the origin and destination (first and last POI respectively) as separate columns with a length of the group's length.
    c. We change the POI column of the group so that the first poi is equal to -2 (as it is a moving point) and replace the POI column with the changed one.
    d. Finally, we horizontally stack the Origin/Destination DataFrame to the original enriched DataFrame.
6. In this final step, we create the unified knowledge base. To do this, we select the columns "oid", "destination", "timestamp", "is_stopped" of the enriched dataset, filter the stopped records, drop the column "is_stopped" and finally group the resulting three-column DataFrame by the values of *oid* and *destination*, using the aggregation of *count* for each group. Now we have a DataFrame that holds information about the number of visits of every unique *oid* to a set of POIs.

Attacks

An example of the Unified Knowledge Base is presented below, used for the examples of the attacks. Let's define EKB as the following:

oid	*poi*	*freq*
1	139	10
1	427	13
1	853	9
1	925	17
2	139	22
2	853	25
2	427	18
3	427	17
3	853	16
3	139	13
3	925	8
4	853	20
4	427	15
4	925	9

With the above background at hand, we define 4 different attack configurations *A*, namely Home and Work, Location, Unique Location, and Location Frequency attack, respectively. In detail:

Home and Work attack: For each individual *oid* in *D*, let *(home, work)* be the pair of top-2 frequent locations visited by the individual and *($freq_{home}$, $freq_{work}$), $freq_{home} > freq_{work}$*, be the pair of their respective frequencies as defined in Eq. 1. If the attack result is not calculated per user, a tuple *(oid, t, loc)* is included in *D'* if *loc* matches either *home* or *work* and the probability of identification prob

(Eq. 2) is over the danger threshold *dt*. Otherwise, if the calculation is performed on users, a user is considered dangerous if at least a tuple *(oid, t, loc)* surpasses the danger threshold *dt*. From the above definition, it turns out that the fewer the individuals that share at least one of their *home* and *work* locations with the target individual, the higher the latter's identification probability. Algorithm 1 (Algorithm Appendix) presents the pseudocode of our Home and Work attack implementation.

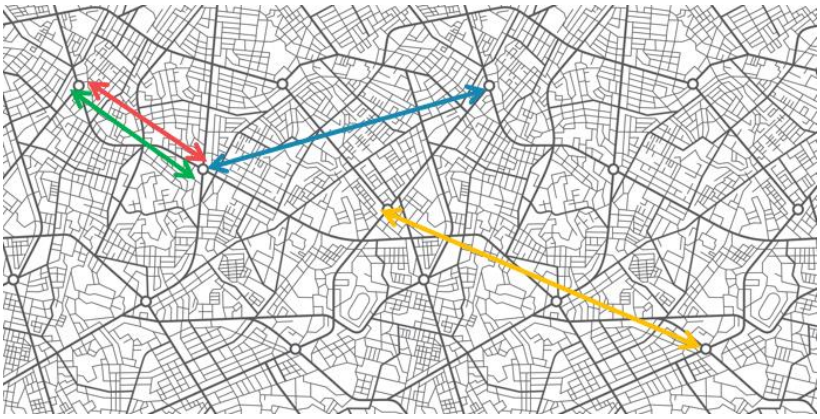

Figure 1: Home and Work Attack example

The execution of the Home and Work Attack on *EKB* produces the following result:

oid	*poi*	*freq*	*prob*
1	925	17	1.0
2	139	22	1.0

Location attack: For each individual *oid* in *D*, let *F* be the set of frequencies of visit per location visited by the individual, calculated as shown in Eq. 1. If the attack result is not calculated per user, a tuple *(oid, t, loc)* is included in *D'* if the probability of identification *prob* (Eq. 2) is over the danger threshold *dt*, the set of matching individuals being those that have the same visit frequency for the location *loc*, as the respective visit frequency in *F*. Otherwise, if the calculation is performed on users, a user is considered dangerous if at least a tuple *(oid, t, loc)* surpasses the danger threshold *dt*. From the above definition, it turns out that the fewer the locations (and the respective number of visits) visited by the rest of the individuals that match the locations (and the respective number of visits) visited by the target individual, the higher the latter's identification probability. Algorithm 2 (Algorithm Appendix) presents the pseudocode of our Location attack implementation.

Figure 2: Location Attack Example

The execution of the Location attack on *EKB* produces the following result:

oid	*poi*	*freq*	*prob*
1	925	17	1.0
1	427	13	1.0
1	925	17	1.0
1	139	10	1.0
1	925	17	1.0
1	853	9	1.0
2	853	25	1.0
2	139	22	1.0
2	853	25	1.0
2	427	18	1.0
2	139	22	1.0
2	427	18	1.0
3	427	17	0.5
3	853	16	0.5
3	427	17	0.5
3	139	13	0.5
3	427	17	1.0
3	925	8	1.0
3	853	16	0.5
3	139	13	0.5
3	853	16	0.5
3	925	8	0.5
3	139	13	1.0
3	925	8	1.0
4	853	20	0.5
4	427	15	0.5
4	853	20	1.0
4	925	9	1.0
4	427	15	1.0
4	925	9	1.0

Unique Location Attack: For each individual *oid* in *D*, let *L* be the set of unique locations visited by the individual. If the attack result is not calculated per user, a tuple *(oid, t, loc)* is included in *D*' if the probability of identification *prob* (Eq. 2) is over the danger threshold *dt*, the set of matching individuals being those that have the same set of unique locations as *L*. Otherwise, if the calculation is performed on users, a user is considered dangerous if at least a tuple *(oid, t, loc)* surpasses the danger threshold *dt*. From the above definition, it turns out that the fewer the sets of unique locations visited by the rest of the individuals that match the set of unique locations visited by the target individual, the higher the latter's identification probability. Evidently, what differentiates Location attack defined earlier from Unique Location attack is that the latter does not take into consideration the number of visits. Algorithm 3 (Algorithm Appendix) presents the pseudocode of our Unique Location attack implementation.

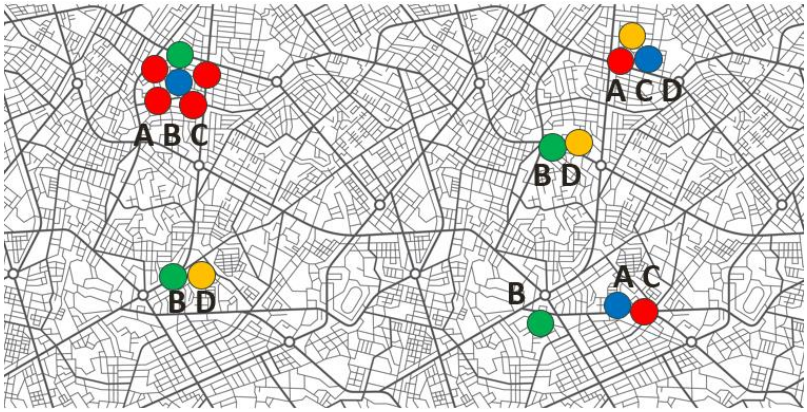

Figure 3: Unique Location Attack example

The execution of the Unique Location attack on *EKB* produces the following result. Notice that the calculation of this attack happens per user only, as it checks the entirety of the sets of locations of users:

oid	*prob*
1	0.5
3	0.5

Location Frequency Attack: For each individual *oid* in *D*, let *F* be the set of frequencies of visit per location visited by the individual, calculated as shown in Eq. 1 and *t* be the tolerance. If the attack result is not calculated per user, a tuple *(oid, t, loc)* is included in *D'* if the probability of identification *prob* (Eq. 2) is over the danger threshold *dt*, the set of matching individuals being those that have a set of approximately equal frequencies of visit for all the locations, compared to the visit frequencies in *F*. Specifically, the frequency needs to have a deviation of $t * freqloc$ from the initial $freq_{loc}$ value for every location *loc*. Otherwise, if the calculation is performed on users, a user is considered dangerous if at least a tuple *(oid, t, loc)* surpasses the danger threshold *dt*. As in the case of Location attack, from the above definition, it turns out that the fewer the locations (and the respective approximated number of visits) visited by the rest of the individuals that match the locations (and the approximated respective number of visits) visited by the target individual, the higher the latter's identification probability. Also, the lower the tolerance *t* value, the higher the target individual's identification probability. Algorithm 4 (Algorithm Appendix) presents the pseudocode of our Location Frequency attack implementation.

Figure 4: Location Frequency Attack example

The execution of the Location Frequency attack on EKB produces the following result:

oid	*poi*	*freq*	*prob*
1	925	17	1.0
1	427	13	1.0
1	925	17	1.0
1	139	10	1.0
1	925	17	1.0
1	853	9	1.0
1	427	13	0.5
1	139	10	0.5
1	427	13	0.5
1	853	9	0.5
1	139	10	0.5
1	853	9	0.5
2	853	25	1.0
2	139	22	1.0
2	853	25	0.5
2	427	18	0.5
2	139	22	1.0
2	427	18	1.0
3	427	17	0.5
3	925	8	0.5
3	853	16	0.5
3	139	13	0.5
3	853	16	0.5
3	925	8	0.5
3	139	13	1.0
3	925	8	1.0
4	853	20	0.5
4	925	9	0.5

Experimental Study

In this section, we present the experimental studies conducted to evaluate the performance of the proposed algorithm against existing methods and the different results that are produced when the parameters of the algorithms are changed.

The experiments were performed on a system with the following specifications: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz, 16GB of RAM. The software environment consisted of Ubuntu 16.04, Rust 1.72.0 and Python 3.10. The datasets used for evaluation are subsets of the Geolife dataset.

The first table (Table 1) shows the comparison of the execution times of the preprocessing procedure and the attacks for the proposed implementation. At the same table, there is a comparison between the execution times of attacks of the proposed and Scikit-mobility implementations. In both the implementations, the attacks are performed per user and not per record. The two implementations are compared on two separate subsets of the Geolife dataset, one with 10^5 and one with 10^6 records.

Table 1: Preprocessing and Attacks execution time comparison

	100k		*1M*	
Procedures	Proposed	Scikit-mobility	Proposed	Scikit-mobility
Preprocessing	0.8	-	10	-
Home and Work	**0.9**	20.7	**1.2**	27.9
Location	**1.7**	69.5	**12.3**	∞
Location Frequency	**1.8**	25	**19.7**	∞
Unique Location	**0.7**	22.9	**7.2**	∞

The results show that for the execution of the exact same attacks, the proposed implementation is by far more efficient than the Scikit-mobility's implementation. It is also worth noting that for the dataset of 1M records, Scikit-mobility didn't manage to produce a result and the execution was forcefully terminated.

The second experiment that was run had as an objective to study the scalability of the execution time of attacks based on the input dataset's size. The following tables (Tables 2, 3, 4 and 5) illustrate the variations in execution times for the attacks, based on the number of records in the datasets utilized. Three distinct datasets were employed for each attack. The primary dataset comprises the initial one million records from the Geolife dataset. Additionally, two subsets of this dataset were used, containing 10^4 and 10^5 records, respectively.

Table 2: Home and Work attack execution times for 3 different input dataset sizes

Dataset size	*Time (sec)*
10k	0.8
100k	1.6
1M	2.05

Table 3: Location attack execution times for 3 different input dataset sizes

Dataset size	*Time (sec)*
10k	11.3
100k	273.6
1M	1305

Table 4: Unique Location attack execution times for 3 different input dataset sizes

Dataset size	*Time (sec)*
10k	2.8
100k	78.6
1M	366

Table 5: Location Frequency attack execution times for 3 different input dataset sizes

Dataset size	*Time (sec)*
10k	11.7
100k	291.1
1M	1376

The final experiment aims to study the effect of the parameter k (in k-anonymity) in the identifiability of individuals. The following table (Table 6) illustrates the percentage of dangerous records detected by the various privacy attacks, based on different danger-detection thresholds. Specifically, we use the thresholds of 0.25 (4-person anonymity), 0.5 (2-person anonymity) and 1.0 (1-person anonymity).

Table 6: Dangerous record percentage based on different danger thresholds

Threshold	*Home & Work*	*Location*	*Location Frequency*	*Unique Location*
1.0	30	100	88	49
0.5	45	100	97	70
0.25	63	100	100	87

Conclusion

The present thesis has focused on the critical junction of data harvesting from urban mobility and privacy, with two related topics: the risks due to privacy attacks and techniques assessing those risks. The mobility data collection, usage, and sharing have significantly increased, especially in scopes like smart cities and location-based services, and so understanding and mitigating privacy risks have become significant concerns. This study reviewed the range of potential privacy attacks that can be performed with mobility data and the sensitivity of information that can be inferred from human trajectories.

These privacy attacks were implemented using the Rust programming language in order to assess the vulnerability of mobility datasets. Such a process allowed for an in-depth assessment of the risks from possible breaches of privacy, providing, in fact, even more detailed information on how adversarial actors can use mobility data to re-identify individuals. Further comparative analysis between the implemented Rust-based tool and the state-of-the-art tool, namely Scikit-mobility Privacy, has shown the potential for improvements in performance, focusing on time efficiency and the accuracy of the results.

The results shown in this thesis demonstrate clearly that continuous innovation is required in the specific domain of privacy-aware mobility analytics. Tools such as Scikit-mobility Privacy represent a great foundation on which to develop a privacy risk assessment, but further development in more performant languages such as Rust could do much toward speeding up these evaluations. Moreover, this work underlines the need for very strong preprocessing techniques together with privacy-preserving methods so that mobility data can safely be used without compromising individual privacy.

Conclusively, this thesis contributes to the broader discussion of mobility data privacy by providing both an implementation and a comprehensive risk analysis of real-world privacy attacks. These results call for further improvement in techniques of privacy preservation and the development of more efficient tools in mobility data analysis by developers, policymakers, and researchers alike. Mobility data will surely increase as smart cities continue to evolve; therefore, it remains a critically active challenge how privacy concerns can be duly safeguarded.

References

[1] M. Sakr, C. Ray, and C. Renso, “Big mobility data analytics: Recent advances and open problems,” GeoInformatica, vol. 26, no. 4, pp.541–549, Oct. 2022, doi: 10.1007/s10707-022-00483-0.

[2] E. Zim´anyi, M. Sakr, and A. Lesuisse, “MobilityDB,” ACM Transactions on Database Systems, vol. 45, no. 4, pp. 1–42, Dec. 2020, doi: 10.1145/3406534.

[3] Z. Fang, L. Chen, Y. Gao, L. Pan, and C. S. Jensen, “Dragoon: a hybrid and efficient big trajectory management system for offline and online analytics,” The VLDB Journal, vol. 30, no. 2, pp. 287–310, Feb. 2021, doi: 10.1007/s00778-021-00652-x.

[4] Q. Yu and J. Yuan, “TransBigData: A Python package for transportation spatio-temporal big data processing, analysis and visualization,” Journal of Open Source Software, vol. 7, no. 71, p. 4021, Mar. 2022, doi: 10.21105/joss.04021.

[5] S. Ruan, R. Li, J. Bao, T. He, and Y. Zheng, “CloudTP: A Cloud-Based Flexible Trajectory Preprocessing Framework,” in 2018 IEEE 34th International Conference on Data Engineering (ICDE), Apr. 2018. http://dx.doi.org/10.1109/icde.2018.00186

[6] S. Haidri, Y. J. Haranwala, V. Bogorny, C. Renso, V. P. da Fonseca, and A. Soares, “PTRAIL — A python package for parallel trajectory data preprocessing,” SoftwareX, vol. 19, p. 101176, Jul. 2022, doi: 10.1016/j.softx.2022.101176.

[7] G. Andrienko, N. Andrienko, F. Giannotti, A. Monreale, and D. Pedreschi, “Movement data anonymity through generalization,” in Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS, Nov. 2009. http://dx.doi.org/10.1145/1667502.1667510

[8] L. Pappalardo, F. Simini, G. Barlacchi, and R. Pellungrini, “scikit-mobility: A Python Library for the Analysis, Generation, and Risk Assessment of Mobility Data,” Journal of Statistical Software, vol. 103, no. 4, 2022, doi: 10.18637/jss.v103.i04.

[9] R. Pellungrini, L. Pappalardo, F. Simini, and A. Monreale, “Modeling Adversarial Behavior Against Mobility Data Privacy,” IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 2, pp. 1145–1158, Feb. 2022, doi: 10.1109/tits.2020.3021911.

Appendix: Algorithms

Algorithm 1 Home and Work Attack algorithm

Input: *KB* = *Dataframe*[*oid,poi,freq*].*sort*(*freq,oid,descending*),*dt* ∈ (0.0,1.0) ⊂Q,*force user* ∈ (*true,false*)

Output: *Dataframe*[*oid,poi,freq,prob*]

```
1: if force user then
2:     O ← Dataframe[oid,prob]
3: else
4:     O ← Dataframe[oid,poi,freq,prob]
5: end if
6: for each oid ∈ oids do
7:     group ← KB[oid = oid]
8:     S ← group[: 2]
9:     if force user then
10:        risk = 0
11:        for instance ∈ S do
12:            matches ← 0
13:            for each oid ∈ oids do
14:                group ← KB[oid = oid][: 2]          // 2 most frequent
15:                if instance.poi ∈ group.pois then
16:                    matches ← matches + 1
17:                end if
18:            end for
19:            prob ← 1/matches
20:            if prob > risk then
21:                risk = prob
22:            end if
23:            if risk ≥ danger threshold then
24:                break
25:            end if
26:        end for
27:        O.stack([instance.oid,prob])
28:    else
29:        OidDataframe ← Dataframe[oid,poi,freq,prob]
30:        for instance ∈ S do
31:            matches ← 0
32:            for each oid ∈ oids do
33:                group ← KB[oid = oid][: 2]          // 2 most frequent
34:                if instance.poi ∈ group.pois then
35:                    matches ← matches + 1
36:                end if
```

```
            end for
            prob ← 1/matches
                OidDataframe.append([instance.oid,instance.poi,instance.freq,prob])
        end for
    end if
    O.stack(OidDataframe)
end for
return O[prob ≥ dt]
```

Algorithm 1 illustrates our method for evaluating the identification risk of the Home and Work Attack for each record or user of a dataset, depending on the execution mode, and keep the records that are considered dangerous. First of all, we create the final DataFrame *O* that will store either the user identification number along with the respective identification probability, if the attack is performed per user, or the records of the input DataFrame along with the respective probability of identification, if the attack is performed per record (lines 1-5). For each unique *oid* in the input DataFrame, we find the two most visited locations (lines 7-8).

If the attack is performed per user, we find the matches for every instance (lines 12-16) and calculate the probability of identification (line 19). If the probability surpasses the current maximum value it is stored as the new value (lines 20-21) and if it surpasses the danger threshold, the individual is considered dangerous, and the procedure stops (lines 23-24). Finally, we append the user identification number and the respective probability to *O* (line 27).

If the attack is performed per record, we find the matches of every instance (lines 31-35). Afterwards, we calculate the respective probability (line 38) and append this to *O* (lines 39 and 42).

Finally, we return the records of *O* that have a probability greater than the danger threshold (line 44) and are thus considered dangerous.

Algorithm 2 Location Attack algorithm

Input: *KB* = *Dataframe*[*oid,poi,freq*].*sort*(*freq,oid,descending*),$k \in [2,+\infty) \subset \mathbb{N}$,$dt \in (0.0,1.0) \subset \mathbb{Q}$

Output: *Dataframe*[*oid,poi,freq,prob*]

```
if force user then
    O ← Dataframe[oid,prob]
else
    O ← Dataframe[oid,poi,freq,prob]
end if
for each oid ∈ oids do
    group ← KB[id = oid]
    S ← group.combinations(k)
    if force user then
        risk = 0
        for instance ∈ S do
            matches ← 0
            for each oid ∈ oids do
                inst ← Dataframe(instance)
                locs_inst ← InnerJoin(KB[id = oid], inst)
                if locs_inst.length 6= inst.length then
                    continue
                else
                    if locs inst[left.freq ≥ right.freq].length != inst.length then
                        continue
                    else
                        matches ← matches + 1
                    end if
                end if
            end for
            prob ← 1/matches
            if prob > risk then
                risk = prob
            end if
            if risk ≥ danger threshold then
                break
            end if
        end for
        O.stack([instance.oid,prob])
    else
        OidDataframe ← Dataframe[oid,poi,freq,prob]
        for instance ∈ S do
            matches ← 0
            for each oid ∈ oids do
```

```
40:                 inst ← Dataframe(instance)
41:                 locs_inst ← InnerJoin(KB[id = oid], inst)
42:                 if locs inst.length != inst.length then
43:                     continue
44:                 else
45:                     if locs inst[left.freq ≥ right.freq].length != inst.length then
46:                         continue
47:                     else
48:                         matches ← matches + 1
49:                     end if
50:                 end if
51:             end for
52:             prob ← 1/matches
53:             OidDataframe.append([instance.oid,instance.poi,instance.freq,prob])
54:         end for
55:     end if
56:     O.stack(OidDataframe)
57: end for
58: return O[prob ≥ dt]
```

Algorithm 2 illustrates our method for evaluating the identification risk of the Location Attack for each record or user of a dataset, depending on the execution mode, and keep the records that are considered dangerous. First of all, we create the final DataFrame *O* that will store either the user identification number along with the respective identification probability, if the attack is performed per user, or the records of the input DataFrame along with the respective probability of identification, if the attack is performed per record (lines 1-5). For each unique *oid* in the input DataFrame, we create a set of instances, which is the combinations of length *k* of all the individual's records (lines 7-8).
If the attack is performed per user, for each instance we perform an Inner Join with the records of other individuals, in order to find the matching *oids* (lines 13-15). We consider an *oid* a match if it has visited all the instance's locations with a greater or equal frequency as the target *oid* (lines 16-20), calculating the total count of those matching *oids* (line 22). Afterwards, we calculate the respective probability (line 26) and if the probability surpasses the current maximum value it is stored as the new value (lines 27-28). If the probability surpasses the danger threshold, the individual is considered dangerous and the procedure stops (lines 30-31). Finally, we append the user identification number and the respective probability to O (line 34).
If the attack is performed per record, we follow the same procedure of performing the Inner Join (lines 40-41), finding the matches and calculating their total count (lines 42-48) and after calculating the respective probability (line 52), we append the instance's rows to O, along with the respective probability (lines 53 and 56).
Finally, we return the records of O that have a probability greater than the danger threshold (line 58), and are thus considered dangerous.

Algorithm 3 Unique Location Attack algorithm

Input: *KB* = *Dataframe*[*oid,poi,freq*].*sort*(*freq,oid,descending*),*k* [2,+∞) ⊂N,*dt* ∈∈ (0.0,1.0) ⊂Q

Output: *Dataframe*[*oid,poi,freq,prob*]

```
1: if force user then
2:         O ← Dataframe[oid,prob]
3: else
4:          O ← Dataframe[oid,poi,freq,prob]
5: end if
6: for each oid ∈ oids do
7:         group ← KB[id = oid]
8:       S ← group.combinations(k)
9:       if force user then
10:          risk = 0
11:          for instance ∈ S do
12:             matches ← 0
13:             instance_pois ← instance.pois
14:              for each oid ∈ oids do
15:                 traj_pois = KB[oid = oid].column(pois)
16:                 sum ← 0
17:                  for each poi ∈ instance pois do
18:                      if traj pois.contains(poi) then
19:                          sum ← sum + 1
20:                      end if
21:                  end for
22:                  if sum = instance pois.length then
23:                      matches ← matches + 1
24:                  end if
25:              end for
26:              prob ← 1/matches
27:              if prob > risk then
28:                  risk = prob
29:              end if
30:              if risk ≥ danger threshold then
31:                  break
32:              end if
33:          end for
34:          O.stack([instance.oid,prob])
35:      else
36:          OidDataframe ← Dataframe[oid,poi,freq,prob]
37:          for instance ∈ S do
```

```
38:          matches ← 0
39:          instance_pois ← instance.pois
40:          for each oid ∈ oids do
                 traj_pois = KB[oid = oid].column(pois)
                 sum ← 0
-----------------------------------------------------------
43:              for each poi ∈ instance pois do
44:                  if traj pois.contains(poi) then
45:                      sum ← sum + 1
46:                  end if
47:              end for
48:              if sum = instance pois.length then
49:                  matches ← matches + 1
50:              end if
51:          end for
52:          prob ← 1/matches
53:                  OidDataframe.append([instance.oid,instance.poi,instance.freq,prob])
54:      end for
55:  end if
56:   O.stack(OidDataframe)
57: end for
58: return O[prob ≥ dt]
```

Algorithm 3 illustrates our method for evaluating the identification risk of the Unique Location Attack for each record or user of a dataset, depending on the execution mode, and keep the records that are considered dangerous. First of all, we create the final DataFrame *O* that will store either the user identification number along with the respective identification probability, if the attack is performed per user, or the records of the input DataFrame along with the respective probability of identification, if the attack is performed per record (lines 1-5). For each unique *oid* in the input DataFrame, we create a set of instances, which is the combinations of length *k* of all the individual's records (lines 7-8).
If the attack is performed per user, for each instance we store the instance's pois (line 13). Subsequently, iterating over the possible matching *oids* (line 14), we iterate over the instance's pois (line 17) to find if the former contains all of them (lines 18-19). If the count is equal to the instance's pois, then the *oid* is considered a match (lines 22-23). Having found all the matching oids, we calculate the respective probability (line 26) and if the probability surpasses the current maximum value it is stored as the new value (lines 27-28). If the probability surpasses the danger threshold, the individual is considered dangerous and the procedure stops (lines 30-31). Finally, we append the user identification number and the respective probability to O (line 34).
If the attack is performed per record, we follow the same procedure of finding the instance's pois (line 39), calculating the total count of matches (lines 43-49) and after calculating the respective probability (line 52), we append the instance's rows to O, along with the respective probability (lines 53 and 56).

Finally, we return the records of O that have a probability greater than the danger threshold (line 58), and are thus considered dangerous.

Algorithm 4 Location Frequency Attack algorithm

Input: *KB* = *Dataframe*[*oid,poi,freq*].*sort*(*freq,oid,descending*),*k* ∈ [2,+∞) ⊂ ℕ ,*dt* ∈ (0.0,1.0) ⊂ℚ, *tolerance* ∈ (0.0,1.0) ⊂ℚ

Output: *Dataframe*[*oid,poi,freq,prob*]

1: **if** force user **then**
2: $O \leftarrow Dataframe[oid,prob]$
3: **else**
4: $O \leftarrow Dataframe[oid,poi,freq,prob]$
5: **end if**
6: **for** each oid ∈ oids **do**
7: $group \leftarrow KB[id = oid]$
8: $S \leftarrow group.combinations(k)$
9: **if** force user **then**
10: $risk = 0$
11: **for** instance ∈ S **do**
12: $matches \leftarrow 0$
13: **for** each oid ∈ oids **do**
14: $inst \leftarrow Dataframe(instance)$
15: $locs \leftarrow KB[id = oid]$
16: $locs\_inst \leftarrow InnerJoin(locs, inst)$
17: **if** locs_inst.length != inst.length **then**
18: continue
19: **else**
20: $condition \leftarrow inst.freq \in [locs.freq * (1 - tolerance), locs.freq * (1 + tolerance)]$
21: **if** locs inst[condition].length != inst.length **then**
22: continue
23: **else**
24: $matches \leftarrow matches + 1$
25: **end if**
26: **end if**
27: **end for**
28: $prob \leftarrow 1/matches$
29: **if** prob > risk **then**
30: risk = prob
31: **end if**
32: **if** risk ≥ danger threshold **then**
33: *break*
34: **end if**
35: **end for**

```
36:            O.stack([instance.oid,prob])
37:        else
38:            OidDataframe ← Dataframe[oid,poi,freq,prob]
39:            for instance ∈ instances do
40:                matches ← 0
41:                for each oid ∈ oids do
42:                    inst ← Dataframe(instances)
43:                    locs ← KB[id = oid]
44:                    locs inst ← InnerJoin(locs,inst)
45:                    if locs_inst.length != inst.length then
46:                        continue
47:                    else
48:                        condition ← inst.freq ∈ [locs.freq ∗ (1 − tolerance),locs.freq ∗ (1 +
    tolerance)]
49:                        if locs inst[condition].length != inst.length then
50:                            continue
51:                        else
52:                            matches ← matches + 1
53:                        end if
54:                    end if
55:                end for
56:                prob ← 1/matches
57:                OidDataframe.append([instance.oid,instance.poi,instance.freq,prob])
58:            end for
59:        end if
60:        O.stack(OidDataframe)
61: end for
62: return O[prob ≥ dt]
```

Algorithm 4 illustrates our method for evaluating the identification risk of the Location Frequency Attack for each record or user of a dataset, depending on the execution mode, and keep the records that are considered dangerous. First of all, we create the final DataFrame O that will store either the user identification number along with the respective identification probability, if the attack is performed per user, or the records of the input DataFrame along with the respective probability of identification, if the attack is performed per record (lines 1-5). For each unique *oid* in the input DataFrame, we create a set of instances, which is the combinations of length *k* of all the individual's records (lines 7-8).

If the attack is performed per user, for each instance we perform and Inner Join with the records of other individuals, in order to find the matching *oids* (lines 14-16). We consider an individual oid_m a match to the target individual oid_t, if he/she has visited all the instance's locations and if each location visit frequency of the target individual is within the range [freq * (1 - t), freq * (1 + t)], where *freq* is the respective frequency of visit for the oid_m (lines 17-24). After finding all the matching *oids* we calculate the respective probability for each

instance (line 28) and if the probability surpasses the current maximum value it is stored as the new value (lines 29-30). If the probability surpasses the danger threshold, the individual is considered dangerous and the procedure stops (lines 32-33). Finally, we append the user identification number and the respective probability to O (line 36).

If the attack is performed per record, we follow the same procedure of performing the Inner Join (lines 42-44), finding the matches and calculating their total count (lines 45-52) and after calculating the respective probability (line 56), we append the instance's rows to O, along with the respective probability (lines 57 and 60).

Finally, we return the records of O that have a probability greater than the danger threshold (line 62) and are thus considered dangerous.