UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Privacy-Preserving Mobility Estimation

Datenschutzfreundliche Mobilitätsabschätzung

Masterarbeit

verfasst am
Institut für IT-Sicherheit

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Max Schulze

ausgegeben und betreut von
Prof. Dr. Esfandiar Mohammadi
Moritz Kirschte

Lübeck, den 15. Juni 2025

IM FOCUS DAS LEBEN

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Max Schulze

Abstract

GTFS data-depend simulations are relatively new. However, they provide insight into a city's transport network. Those insights are helpful for simulating user behaviours accurately. For this, we utilize GTFS data and by combining the A*-algorithm with the Haversine distance, we implement a simulation correctly mimicking real-world behaviour.

Privacy-Preserving Mobility Estimation (MobiTrie) is a novel framework for releasing accurate mobility patterns under ε-differential privacy. MobiTrie first traverses given trajectories to create rolling 3-grams, preserving subsequences and capturing local transfers. Additionally, the universe of all possible 3 grams is taken into account, ensuring a real-world adaptation. It then builds a ε-differentially private 3-gram prefix tree by adding Laplace noise to the frequencies of all 3-grams and then selecting the most significant 3-grams based on a random threshold. The selected 3-grams are then assembled into a trie using an F1-score-based rejection sampling process, providing a quality measurement. We evaluated $10,000$ and $1,000,000$ simulated users on Berlin's public transportion network. Additionally, we evaluate the MSNBC dataset [15] for a better comparability with Chen et al. [6]. MobiTrie achieves high replayability (fitness = 0.96), strong pattern recovery ($F_1 = 82$), and low subsequence error rates (≈ 0.26 to 0.05) even at tight privacy budgets ($\varepsilon = 0.1$). These results demonstrate that MobiTrie provides actionable mobility insights while provably protecting individual privacy, offering a practical solution for the release of mobility data.

Zusammenfassung

Datenabhängige GTFS-Simulationen sind relativ neu. Sie bieten jedoch Einblicke in das Verkehrsnetz einer Stadt. Diese Einblicke sind nützlich, um Nutzerverhalten genau zu simulieren. Zu diesem Zweck verwenden wir GTFS-Daten und kombinieren den A*-Algorithmus mit der Haversine-Distanz, um eine Simulation zu implementieren, die das Verhalten der realen Welt korrekt nachahmt.

MobiTrie (Privacy-Preserving Mobility Estimation) ist ein neuartiger Rahmen für die Freigabe genauer Mobilitätsmuster unter ε-differentieller Privatsphäre. MobiTrie durchläuft zunächst gegebene Trajektorien, um rollende 3-Gramme zu erstellen, wobei Teilsequenzen erhalten bleiben und lokale Umstiege erfasst werden. Zusätzlich wird das Universum aller möglichen 3-Gramme berücksichtigt, um eine Anpassung an die reale Welt zu gewährleisten. Anschließend wird ein ε-differentiell privater 3-Gramm-Präfixbaum erstellt, indem Laplace-Rauschen zu den Frequenzen aller 3-Gramme hinzugefügt wird und dann die signifikantesten 3-Gramme basierend auf auf einem zufälligen Schwellenwert ausgewählt werden. Die ausgewählten 3-Gramme werden dann mit Hilfe eines F1-Score-basierten Ablehnungsstichprobenverfahrens zu einem Trie zusammengefügt, was eine Qualitätsmessung ermöglicht. Mit unserer Simulation haben wir 10.000 und 1.000.000 Nutzer im öffentlichen Verkehrsnetz Berlins simuliert und für die Evaluation von *MobiTrie* genutzt. Zusätzlich verwenden wir den MSNBC-Datensatz [15], um eine bessere Vergleichbarkeit mit Chen et al. [6], die zu erreichen. MobiTrie erreicht eine hohe Fitness von 0.96, einen hohen F_1 Wert $> 0,82$ und niedrige Teilsequenzfehlerraten ($\approx 0,26 - 0,05$) selbst bei knappen Datenschutzbudgets ($\varepsilon = 0,1$). Diese Ergebnisse zeigen, dass MobiTrie verwertbare Erkenntnisse über die Mobilität liefert und dabei nachweislich die Privatsphäre eines Einzelnen schützt, und damit eine praktische Lösung für die Veröffentlichung von Mobilitätsdaten liefert.

Acknowledgements

I would like to thank Moritz Kirschte for his extensive support during the work and for answering all my questions. Thanks also to Esfandiar Mohammadi, who made this work possible and also strongly supported me. Special thanks to Lisa who supported me throughout my studies and during the intense process of the thesis.

Contents

1 Introduction **1**
1.1 Contribution 2
1.2 Structure 2

2 Related Work **5**

3 Preliminaries **9**
3.1 Mobility Data Processing 9
3.1.1 Trajectories 9
3.1.2 Trajectory Dataset 9
3.1.3 Prefix Tree 10
3.1.4 N-Gram for Trajectories 11
3.2 Differential Privacy 13
3.2.1 Laplace Mechansim 14
3.2.2 Rejection Sampling 14

4 Problem Statement **17**
4.1 Motivation 17
4.1.1 Limits of Existing Data and Methods 17
4.1.2 Our Approach 19
4.2 Challenges 19
4.3 Threat Models 20

5 Simulating Mobility Data **22**
5.1 Idea 22
5.2 Pre-Process Data 23
5.3 Hotspot Generation 23
5.4 Haversine Distance 24
5.5 A* Algorithm 26
5.6 Generating the Simulation 28

6 MobiTrie (Mobility Prefix-Tree) **31**
6.1 Idea 31
6.2 Pre-Processing Trajectories 31
6.2.1 Advantages of Rolling 3-Grams 32

6.2.2 Constructing Rolling 3-Grams 33
6.3 Processing 3-Grams 33
6.4 Contructing the Trie 34
6.4.1 Possible Tries 36
6.4.2 Privacy Guarantee 37

7 Experiments **43**
7.1 Research Questions 43
7.2 Experimental Setup 43
7.3 Experimental Results - Performance Metrics 45
7.3.1 Detailed Evaluation 46
7.4 Experimental Results - Mean Error Rate 49
7.4.1 Detailed Evaluation 50
7.5 Anserwering the Research Questions 52

8 Conclusion **55**
8.1 Future Work 56

Bibliography **58**

A Figures **iii**
A.1 Figure of the Problem Statement iv
A.2 Figures of the Simulation v
A.3 Evaluation Results - Mean Error Rate viii

B Possible Tries on MSNBC **xii**

List of Figures

Figure 3.2 Prefix-tree of the trajectories present in Table 3.1. • represents the end of a trajectory. 12

Figure 3.3 Prefix-trees for 2-gram, 3-gram, and 4-gram sub-paths of the example trajectory dataset in Table 3.1. 13

Figure 3.4 Example figure for the Laplace noise distribution. The closer one gets to the mean of the distribution, the more noise is added. 15

Figure 4.1 A heat-mapped trajectory graph of Berlin's transit network. Star markers show Endpoints. The path colour gets red the more users take this path. 18

Figure 5.1 Differnence between a completely random simulation (Figure 5.1b) and when using weighted randomness through the hotspot generation (Figure 5.1a). For better readability of those figures see Section A.2. 25

Figure 5.2 Illustration of the Great-Circle distance of two points (P and Q) on a sphere [24]. The points u and v are antipodal points. 26

Figure 5.3 Simulation of $100,000$ users taking the public transport system in Berlin, displayed as a heatmap. The more a path is taken, the larger and redder the connection is displayed. The trajectories are computed using the A* algorithm, with the Haversine distance as the heuristic function. Appendix A.2 shows the figures in full size. 29

Figure 6.1 Trie with rolling- and static 3-grams, based on Table 3.1 on page 10. Those two tries represent the same data, with different 3-gram approaches. 32

Figure 6.2 Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.1$. 37

Figure 6.3 Possible trie of the MSNBC dataset [15], using $\varepsilon = 1.0$. 37

Figure 6.4 Baseline trie of the MSNBC dataset [15]. The baseline indicates that no differential privacy mechanisms were employed in its creation. 37

Figure 7.2 Evaluation results of the F1 score, fitness, precision, and recall. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value over all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is 1.0 for each of those metrics. 46

Figure 7.3 Evaluation results of the Accuracy, Jaccard index, and the FNR. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value over all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is 1.0 for the Accuracy and the Jaccard index. The grey line shows the baseline for the FNR, which is 0.0. 47

Figure 7.4 TP, FP, FN, and TN evaluation results. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value for all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is $11,044$ for the TP. The green line indicates the addition of TP and FN, showing that they reach the baseline value and include all elements. 48

Figure 7.8 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = \{0.1, 0.2, 0.5, 1.0\}$, 100 times for each value. The results show the mean and standard deviation of those 100 runs. See Appendix A.3 for all plotted results. For visibility reasons, the results for $\varepsilon = 0.8$ can be found in Append A.3. 51

Figure A.1 A heat-mapped trajectory graph of Berlin's transit network. Star markers show endpoints. The path color gets redder the users take this path. iv

Figure A.2 Simulation using the hotspot generation from Algorithm 3 on page 24. Here in fullsize, for better readability. v

Figure A.3 Simulation without using the hotspot generation from Algorithm 3 on page 24. Here in fullsize, for better readability. vi

Figure A.4 Simulation of 100,000 users taking the public transport system in Berlin, displayed as a heatmap. The more a path is taken, the bigger and darker the connection is displayed. The trajectories are computed using the A* algorithm, with the Haversine distance as the heuristic function. vii

Figure A.5 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.1$ The result shows the mean and standard deviation of 100 runs. viii

Figure A.6 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.2$ The result shows the mean and standard deviation of 100 runs. viii

Figure A.7 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.5$ The result shows the mean and standard deviation of 100 runs. ix

Figure A.8 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.8$ The result shows the mean and standard deviation of 100 runs. ix

Figure A.9 Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.0$ The result shows the mean and standard deviation of 100 runs. x

Figure B.1 Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.1$. xiii

Figure B.2 Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.5$. xiv
Figure B.3 Possible trie of the MSNBC dataset [15], using $\varepsilon = 1.0$. xv
Figure B.4 Baseline trie of the MSNBC dataset [15]. Baseline means that no differential privacy mechanisms where used in the creation. xvi

List of Tables

Table 3.1 Example trajectory dataset $\mathcal{D}$, on locations L_1 to L_7. 10

Table 7.1 Experimental dataset characteristics. 44

Table 7.5 Mean values of the performance metrics at different ε budgets, for $10,000$ simulated users. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us. 49

Table 7.6 Mean values of the performance metrics at different ε budgets, for $1,000,000$ simulated users. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us. 49

Table 7.7 Mean values of the performance metrics at different ε budgets, for the MSNBC dataset [15]. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us. 49

Table 7.9 Relative mean error rate by subsequence lengths for various ε budgets, for $10,000$ simulated users (Algorithm 5). For each ε budget, with increasing `subset_length` a decrease of the `mean_error` is expected. 52

Table 7.10 Relative mean error rate by subsequence lengths for various ε budgets, for $1,000,000$ simulated users (Algorithm 5). For each ε budget, with increasing `subset_length` a decrease of the `mean_error` is expected. 53

Table 7.11 Relative mean error rate by subsequence lengths for various ε budgets, for the MSNBC dataset. With increasing `subset_length` the `mean_error` decreases. 53

1

Introduction

The analysis of movement patterns is a rapidly growing area as more users utilise public transport systems and fitness apps to track their movements through GPS coordinates and other trackers, such as those used to locate keys. All of those usages reveal insights into the behaviours of individuals, such as their homes and workplaces. Nonetheless, those insights are of special interest to the fine-grained analytics of urban traffic, transport system planning, and smart-city services. However, as adversaries have repeatedly demonstrated, even after stripping identifiers, unique or low-frequent movement patterns can re-identify individuals, exposing sensitive information such as home and work locations. This discrepancy between utility and privacy has led to research analysing how to publish mobility data without violating the privacy of individuals.

As mobility data poses the inherent risks of re-identifying individuals [3], transport system providers do not make this data publicly available. However, such data is often of considerable interest as researchers may want to prove that their implementation works and is not revealing crucial information. Due to the lack of datasets, we implement a real-world simulation (Chapter 5) based solely on GTFS data. As this data is publicly available, it is beneficial. By utilising the A*-algorithm with the Haversine Using distance as the heuristic function, we ensure that the simulated data is as natural as possible.

One of the first research fields in this area is the approximation of classical k-anonymity so that the published data is sanitised based on generalisation and the assumption that an adversary only has limited background knowledge. However, as proven by [18], limited knowledge about an individual is sufficient to break k-anonymity. Due to this, differential privacy has been considered a better approach to privatising datasets, as this gives more robust and quantifiable privacy guarantees. Nevertheless, current work utilises differential privacy and often relies on clustering, stripping away crucial information and leading to a loss of utility. One approach similar to ours, proposed by Chen et al. [6], suffers from utility loss as movements are introduced, which would never happen in the underlying transport network. Furthermore, Chen et al. treat the data given as the complete universe of stops, which holds for their proof but fails in practice. In real-world scenarios, any given dataset typically represents a subset of the whole network, so it omits some stops altogether. An adversary can exploit this by considering two neighbouring datasets $D \sim D'$. If D' includes a stop absent from D, Chen et al.'s mechanism does not know that stop. Consequently, the adversary can distinguish whether that stop was present in $\mathcal{D}$ or not, violating privacy.

This leads to the implementation of the *MobiTrie* algorithm. The algorithm's goal is to provide provable ε-differential privacy while preserving the dataset's utility and not introducing impossible movements. For this, we use Laplace noise [12] and the rejection sampling process [17] to generate prefix trees that are private and considered good. For the building process, we use the concept of rolling n-grams, which preserves overlapping subsequences of length n, to capture local transfers and longer movements. Additionally, all possible 3-grams from the underlying transport network are taken into account. This ensures that our privacy assumptions hold in real-world scenarios, improving the assumption of Chen et al. [6]. By creating a union of the given datasets 3-grams and the universe of all possible 3-grams, we get a future-proof set of (3-gram, count) tuples. Since we noise the count of all n-grams in the union set and only the most significant ones are selected based on a random threshold, all n-gram prefix trees generated are proven to fulfill ε-differential privacy while the utility of the dataset is preserved.

We evaluate *MobiTrie* on a realistic simulation of Berlins public transport system network by simulating $10,000$ and $1,000,000$ users, and the MSNBC dataset [15], used by Chen et al. [6], for a better comparability. The results show that even for a substantial privacy budget of $\varepsilon = 0.1$ the resulting trie achieves high replayability (fitness$= 0.9$) and a reasonable F1-score (> 0.51) on the MSNBC dataset. The relative error rate of variable length subsequences is within 0.26 to 0.05, outperforming [6], at a length of 20. These results demonstrate that *MobiTrie* supports valuable mobility analysis, such as hotspot detections and route profiling, while providing provable ε-differential privacy against adversaries.

1.1 Contribution

The contribution of this thesis is as follows:

1. We implement a realistic simulation of Berlin by using publicly available GTF files.
   - We utilise the A* algorithm, with the Haversine distance as a heuristic function, to provide realistic paths.
   - We define weighted stops as destinations to enhance the realism of the simulation.
2. We define rolling n-grams to preserve overlapping subsequences, capturing local transfers and longer movements.
3. We utilise the universe of all 3-grams possible in the underlying transport network for provable ε-differential privacy.
4. We construct a differentially private trie by noising frequencies of 3-grams via the Laplace mechanism and selecting the most significant by a random threshold between 0.0 and $\frac{20\cdot\sqrt{2}}{\varepsilon}$. All tries are assembled through an F1-score-based rejection sampler, providing a quality measurement.

1.2 Structure

Chapter 2 gives insights about work that is related to us. Terms and algorithms used throughout this thesis are explained in chapter 3. In chapter 4, we discuss the motivation behind this thesis and the threat models we consider. Chapter 5 looks at the idea behind the simulation based on GTF files and how it is implemented using known techniques. In chapter 6 the idea behind the ε-differentially private *MobiTrie* algorithm and how we implemented it is discussed. The utility and error rate evaluation results are discussed in chapter 7. Finally, in chapter 8, we conclude the thesis and look at future work.

2

Related Work

Differential privacy for mobility data is a rapidly growing area, but most approaches have critical shortcomings alongside their innovations. Early sequence-based frameworks, such as [1, 2, 8, 16, 27] have published privacy-preserving trajectory data methods.

Abul et al. [1] propose a (k, δ)-anonymity method for publishing trajectories. (k, δ)-anonymity extends the classical k-anonymity to moving objects, by adding some uncertainty δ. They assume the inherent inconsistencies of GPS data of positioning systems, where δ represents this imperfection. By doing so, the trajectories are modified by space translation so that k different trajectories co-exist in a cylinder of radius δ.

Andrienko et al. [2] present an approach based on generalization to achieve k-anonymity. Their generalization scheme depends on the underlying trajectory dataset rather than a fixed grid hierarchy. By replacing exact locations in each trajectory with approximate locations, k-anonymity is achieved, aiming to hide locations using a generalization.

Chen et al. [8] consider that the knowledge of an adversary on a target victim is bounded by L location-time doublets, leading them to the $(K, C)_L$-privacy model. This model mitigates identity linkages on trajectory data by a $\leqslant 1/K$ probability and attribute linkages via trajectory data by a $\leqslant C$ probability.

Hu et al. [16] discuss the problem of releasing a k-anonymized trajectory dataset alongside a sensitive event dataset, where at least k users from the trajectory dataset share every event from the event dataset. As an adversary can find spatiotemporal coincidences in those linked datasets, they propose a new approach to generalise user locations concerning the event dataset. As long as the adversary has no background knowledge, he can only infer that each of the k users has the same $1/k$ probability of being the genuine user.

Yarovoy et al. [27] propose a novel framework to publish a moving-object database (MOB) under provable k-anonymity, even though different objects may expose location information at distinct ”quasi-identifiers”. As they prove that known k-anonymity is insufficient, they propose a symmetric k-anonymity scheme. It ensures that every individual node must have degree $\geqslant k$ and each edge must be bi-directional, ensuring that no matching can single out a specific trajectory.

All of the above approaches [1, 2, 8, 16, 27] propose and build schemes based on partition-based privacy (k-anonymity), which is not a sufficient protection against adversaries with unlimited background knowledge. This is why we propose an algorithm that utilizes ε-differential privacy, which has been proven to provide more robust privacy guarantees.

Hierarchical spatial clustering, as in [4, 14], reduces the domain complexity by mapping coordinates into multi-resolution grids or Hilbert-curve clusters before building a differentially private tree. Although these proposed approaches increase accuracy by clustering nearby locations, they cancel out granular movements, which can mask important movement patterns, such as short bus trips.

Markov-chain-based approaches, such as PrivTrace [22] and DPTraj-PM [23], add noise to transition counts and generate synthetic trajectories by sampling from a noisy chain. While these methods exploit sequential patterns are better than fixed-depth trees, but high-order Markov models still require estimating large transition matrices, which suffer from noise blow-up as the number of states increases. DPTraj-PM uses prefix trees and clustering to decrease the number of states. However, it still suffers from clustering, providing inexact results for short trips. Nevertheless, it shows that key movement patterns are preserved.

Local DP (LDP) methods [10, 28] eliminate the trust requirement towards a central authority by having each user perturb their trajectories locally before publishing. Cunningham et al. [10] proposes hierarchical n-grams perturbing user trajectories by breaking each trajectory into rolling n-grams and then applying LDP noise to each n-gram. Those n-grams are organized in a multi-level space-time-category (STC) hierarchy, where each level groups the trajectory into region-level cells that capture spatial and temporal proximity. Zhang et al. [28] propose a complementary pivot-sampling approach that exploits the directional relationships in trajectories, reducing the perturbation domain. Each trajectory is duplicated into two interleaved copies (even and odd indices), and pivots in each copy are perturbed via the exponential mechanism. While both approaches ensure ε-LDP, the amount of noise needed is significantly larger, and both need external information about POIs (points of interest), which might not always be given.

Aggregated release models, as proposed by Haydari et al. [13] and Shaham et al. [21], noise coordinates and flows to achieve ε-differential privacy. Haydari et al. [13] release an aggregated flow graph by noising the coordinates of a trajectory and then snapping those noised points to a road graph, returning a noise link-trajectory count for each edge. Shaham et al. [21] view trajectories as multi-dimensional histograms. Those histograms are partitioned into cells by data density, and noise is added optionally. While both methods provide provable ε differential privacy, they cannot support fine-grained sequence patterns and may obscure crucial information.

Two approaches close to ours are [7, 6], as they use prefix trees. Chen et al. [7] build prefix trees by noising each node in the trie while enforcing count consistency. This approach preserves path patterns. However, as more locations are added, the tree depth increases significantly, ultimately increasing the noise needed. This problem is faced in [6] by introducing variable n-gram prefix trees. For which subsequences of length n are extracted into a noisy n-gram tree to synthesize trajectories. While this approach reduces the domain size compared to counting full routes, it suffers from semantic inconsistencies as routes not possible are constructed. By adding all possibilities to all prefixes, it produces routes that never occur in the transport network, introducing errors that decrease the utility.

Moreover, the prefix trees constructed in both works only consider the given dataset as the whole universe. This limitation overlooks the transportation network, allowing an adversary to exploit a neighbouring dataset. By inserting a stop that exists in the actual network but not in the observed dataset, they could infer the presence or absence of specific data points, undermining privacy guarantees.

While most existing approaches fulfill their considered privacy guarantees, some are not applicable as they assume restricted adversaries by the use of k-anonymity [1, 2, 8, 16, 27]. Other approaches using differential privacy, on the other hand, suffer from clustering, which strips important information or has noise scaling with the number of locations added. While [7, 6] propose prefix trees and n-grams, they suffer from the possibility of creating non-deterministic paths. Therefore, the main contribution of this work is to implement a 3-gram prefix tree, which preserves the infrastructure of the dataset and fulfills ε-differential privacy while maintaining a good utility for further analysis. Importantly, our method integrates the full transportation network into the generation process, mitigating vulnerabilities from external knowledge attacks and enhancing overall privacy robustness.

3

Preliminaries

This chapter focuses on the two main topics of this thesis: mobility data and differential privacy. First, we will provide an overview of mobility data and its processing. Afterwards, we will define differential privacy and examine the relevant theorems for this thesis.

3.1 Mobility Data Processing

The most essential parts are analysing and visualising mobility data, trajectories, datasets, and prefix trees. In the following, these concepts are formally defined and described to give a better overview of what is done throughout this thesis.

3.1.1 Trajectories

Trajectories describe the path taken by any person. For this, let $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$ be the universe of locations, with $|\mathcal{L}|$ being the length of the universe, which corresponds to the total number of locations in $\mathcal{L}$. Depending on the use case, locations can be anything from train stations to coffee shops. In this thesis, we focus on public transport stations as the location universe. A trajectory is an ordered list of locations from $\mathcal{L}$.

Definition 3.1 (Trajectory).
A trajectory $\mathcal{T}$ of length $|\mathcal{T}|$ is an ordered list $\mathcal{T} = [L_1, L_2, L_3, \dots, L_{|\mathcal{T}|}]$ where each location L_i belongs to the set of locations $\mathcal{L}$, with $\forall i \in \{1, 2, \dots, |\mathcal{T}|\}, L_i \in \mathcal{L}$.

Locations in a trajectory may be repeated, but the same location cannot appear consecutively. For instance, the trajectory $[L_1, L_2, L_3, L_2]$ is valid, whereas the trajectory $[L_1, L_2, L_2]$ is not.

3.1.2 Trajectory Dataset

One trajectory is not sufficient to compute the mobility of multiple users. Therefore, we utilise a trajectory dataset comprising multiple trajectories.

Definition 3.2 (Trajectory Dataset).
A trajectory dataset $\mathcal{D}$ of size $|\mathcal{D}|$ is a multi-set of trajectories $\mathcal{D} = \{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_{|\mathcal{D}|}\}$,

where each $\mathcal{T}_i$ is a trajectory as defined in Definition 3.1 on the previous page. As $\mathcal{D}$ is a multi-set, each trajectory may occur more than once.

Trajectory	Path
$\mathcal{T}_1$	L_1, L_3, L_4, L_6
$\mathcal{T}_2$	L_2, L_3, L_4, L_5, L_7
$\mathcal{T}_3$	L_1, L_3, L_4
$\mathcal{T}_4$	L_3, L_4, L_5, L_7
$\mathcal{T}_5$	L_2, L_4, L_6
$\mathcal{T}_6$	L_1, L_3, L_4, L_5, L_7
$\mathcal{T}_7$	$L_1, L_3, L_4, L_5, L_7, L_1$
$\mathcal{T}_8$	L_2, L_3, L_4, L_5
$\mathcal{T}_9$	L_5, L_7, L_1, L_3

Table 3.1: Example trajectory dataset $\mathcal{D}$, on locations L_1 to L_7.

Table 3.1 shows an example dataset of trajectories. As seen in Table 3.1, the number of trajectories $\mathcal{T}_i$ is greater than the number of locations L_i, which is a common observation because thousands to millions of users are observed. Each trajectory corresponds to one observation. Therefore, one user might have more than one entry in the dataset. For all trajectory datasets, as defined in Definition 3.2 on the previous page, holds Definition 3.3.

Definition 3.3 (Size–Location Inequality).
Let $\mathcal{D}$ be a trajectory dataset as in Definition 3.2, and let

$$|\mathcal{D}| = \text{number of trajectories in } \mathcal{D}, \qquad \mathcal{L} = \{\text{all distinct locations appearing in } \mathcal{D}\}.$$

Then $\mathcal{D}$ satisfies the *size–location inequality* if

$$|D| > |\mathcal{L}|.$$

This applies to all trajectory datasets considered (cf. Table 3.1).

Definition 3.3 is essential to obtain reasonable location counts. Otherwise, the count results are not suitable for noising, contradicting differential privacy (see Section 3.2 on page 13).

3.1.3 Prefix Tree

Several options exist to visualise trajectory datasets $\mathcal{D}$, but prefix trees are a common representation that ensures low overhead. These trees group all trajectories that share the same prefix into the same branch, enabling efficient visualisation and querying.

Definition 3.4 (Location Prefix).
Let $\mathcal{T} = [L_1, L_2, \ldots, L_{|\mathcal{T}|}]$, be a trajectory over the location universe $\mathcal{L}$. A location $L_i \in \mathcal{T}$ is prefix to a location $L_j \in \mathcal{T}$, if

$$L_i \prec_{\mathcal{T}} L_j \quad \Longleftarrow \quad 1 \leqslant i < j \leqslant |\mathcal{T}|.$$

Based on Definition 3.4, considering a trajectory $\mathcal{T}_1 = [L_1, L_2, L_5, L_4]$, L_1 is a prefix of L_2, but not of L_5. Therefore, when looking at one trajectory, each location can only have one prefix. However, considering a trajectory dataset $\mathcal{D}$, each location can have more than one prefix, formally defined below as a *Location Prefix-Set.*

Definition 3.5 (Location Prefix-Set).
Given a trajectory dataset $\mathcal{D}$ of length $|\mathcal{D}|$ over a location universe $\mathcal{L}$. For each location $L \in \mathcal{L}$, its *prefix-set* is defined as

$$\mathcal{P}(L) = \{L' \in \mathcal{L} : \exists\, \mathcal{T} \in \mathcal{D},\ \ L' \prec_{\mathcal{T}} L\}.$$

That is, $\mathcal{P}(L)$ collects all locations that ever occur before L in any trajectory $\mathcal{T} \in \mathcal{D}$.

We can construct a prefix tree using Definitions 3.4 and 3.5. A prefix tree is a compact representation of a trajectory dataset $\mathcal{D}$, as it groups all trajectories of a prefix set $\mathcal{P}_i$ into the same branch. Allowing efficient querying and visualisation of the dataset. With this, each location is a node, and the edges represent the prefixes of the location. Therefore, an edge represents the movement from L_i to L_j. Prefix trees are formally defined as follows.

Definition 3.6 (Prefix Tree).
Let $\mathcal{D}$ be a trajectory dataset over a location universe $\mathcal{L}$. Denote $\mathcal{P}(L) = \{L' \in \mathcal{L} : \exists\, \mathcal{T} \in \mathcal{D},\ \ L' \prec_{\mathcal{T}} L\}$ (Definition 3.5), as the *prefix-set* of each location L. We define the prefix tree $\mathcal{PT} = (V, E, root(\mathcal{PT}))$ as:

$$\begin{aligned} V &= \{\, p \in \mathcal{L}^* : \exists\, \mathcal{T} \in \mathcal{D},\ p \text{ is a prefix of } \mathcal{T}\}, \\ root(\mathcal{PT}) &= \sigma, \\ E &= \{\, (p,\ p \cdot L) \in V \times V : L \in \mathcal{L},\ p \cdot L \text{ is a prefix of some } \mathcal{T} \in \mathcal{D}\}. \end{aligned}$$

That is, V contains every sequence of locations that begins any trajectory $\mathcal{T} \in \mathcal{D}$. We then add edges E so that each edge connects a prefix to all its suffixes.

The following holds:

– The root σ sits at depth 0.
– A node $p \in V$ sits at depth $|p|$, the number of locations in p.
– A node p is a *leaf* exactly if no extension $p \cdot L$ lies in V—equivalently, p is a complete trajectory in $\mathcal{D}$.

3.1.4 N-Gram for Trajectories

N-grams are tuples of locations of size n. Each location in the tuple is from the same trajectory, and the order of the locations in the tuple corresponds to the order of the corresponding trajectory.

Definition 3.7 (N-Gram).
Given a trajectory $\mathcal{T}$, following Definition 3.1, a n-gram $\mathcal{G}_n(\mathcal{T})$ of size n is an ordered tuple $\mathcal{G}_n(\mathcal{T}) = (L_1, L_2, \ldots, L_n)$, where each location L_i is drawn from $\mathcal{T}$, and the order of L_i in $\mathcal{G}_n(\mathcal{T})$ corresponds to its order in $\mathcal{T}$. Formally, for a trajectory $\mathcal{T} = [L_1, L_2, \ldots, L_{|\mathcal{T}|}]$, $\mathcal{G}_n(\mathcal{T})$ represents the first n locations in $\mathcal{T}$.

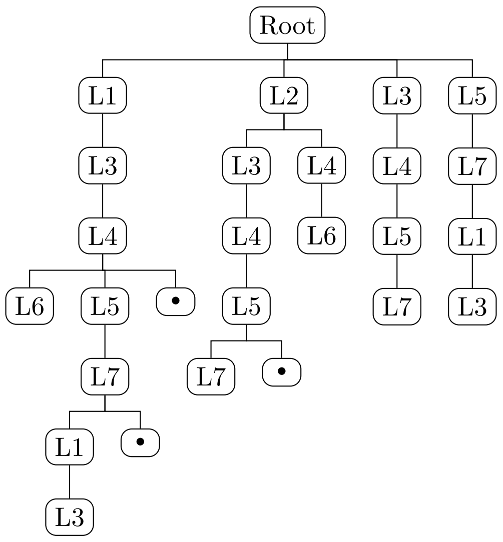


Figure 3.2: Prefix-tree of the trajectories present in Table 3.1. • represents the end of a trajectory.

Looking at Table 3.1 on page 10, $\mathcal{G}_3(\mathcal{T}_1) = (L_1, L_3, L_4)$ and $\mathcal{G}_3(\mathcal{T}_4) = (L_3, L_4, L_5)$ are valid 3-grams for trajectories $\mathcal{T}_1$ and $\mathcal{T}_4$, but $\mathcal{G}_3(\mathcal{T}_j) = (L_3, L_5, L_6)$ is for no trajectory valid, as $\mathcal{T}_j \notin \mathcal{D}$.

Using n-grams, we can construct a prefix tree, as defined in Definition 3.6. The depth of the prefix tree scales with the length of the n-grams used.

Definition 3.8 (N-Gram Prefix-Tree (Trie)).
Let $\mathcal{D}$ be the trajectory dataset of length $|\mathcal{D}|$, where each trajectory $\mathcal{T}_i$ is a finite sequence of locations drawn from the universe $\mathcal{L}$. Denote by

$$\mathcal{G}_n(\mathcal{T}_i) = \{(s_k, s_{k+1}, \dots, s_{k+(n-1)}) : (s_1, \dots, s_{|\mathcal{T}_i|}) = \mathcal{T}_i,\ 1 \leqslant k \leqslant |\mathcal{T}_i| - (n-1)\}$$

the set of all contiguous n-grams in $\mathcal{T}_i$, and let $\mathcal{G}_n(\mathcal{D}) = \bigcup_{i=1}^{|\mathcal{D}|} \mathcal{G}_n(\mathcal{T}_i)$.

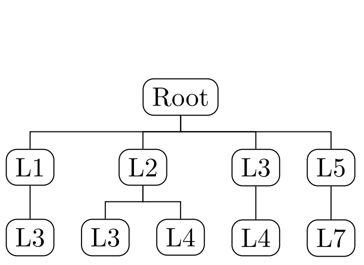
The n-gram prefix-tree (trie), $\mathcal{GT} = (V, E, root)$ is defined by

$$\begin{aligned} V &= \{\, p \in \mathcal{L}^* : \exists\, g \in \mathcal{G}_n(\mathcal{D}),\ p \text{ is a prefix of } g \}, \\ root &= \sigma, \\ E &= \{\, (p,\, p \cdot L_j) \in V \times V : L_j \in \mathcal{L},\ |p| < n \}. \end{aligned}$$

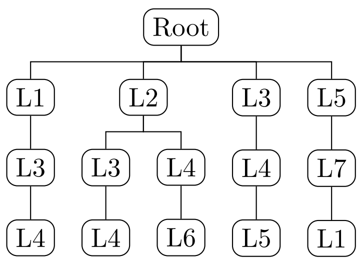
The following holds:

- Each node $p \in V$ sits at depth $|p|$, the length of the prefix.
- The root σ is at depth 0.
- Leaves correspond exactly to the full n-grams in $\mathcal{G}_n(\mathcal{D})$.

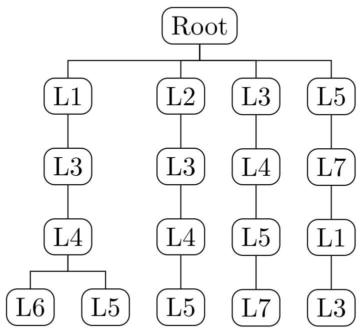
Based on Table 3.1 on page 10 and Definitions 3.7 and 3.8 tries $\mathcal{GT}$ of different n-gram lengths, would look like the following:



(a) 2-gram prefix tree



(b) 3-gram prefix tree



(c) 4-gram prefix tree

Figure 3.3: Prefix-trees for 2-gram, 3-gram, and 4-gram sub-paths of the example trajectory dataset in Table 3.1.

3.2 Differential Privacy

Differential privacy aims to hide the influence of a single data point on a resulting model when analysing it. Therefore, it should not be possible for an adversary or any other individual to gain enough information from the model to be sure about the absence or presence of a single data point.

More formally, it is assumed that an adversary has access to two neighbouring datasets $\mathcal{D}$ and $\mathcal{D}'$, meaning that $\mathcal{D}$ and $\mathcal{D}'$ differ in at most one element. We assume that the adversary has access to the output model of M and two neighbouring datasets $\mathcal{D}$ and $\mathcal{D}'$. The goal of the adversary is now to identify whether $\mathcal{D}$ or $\mathcal{D}'$ was the input dataset.

Two needed parameters to define the privacy guarantees M can give are ε and δ. While ε defines the maximum difference between multiple outputs of M, δ describes the probability that elements are not covered by ε. It is important to note that the lower the ε value, the higher the noise added and, therefore, the privacy gained. However, also the lower the accuracy of the output model $M(\mathcal{D})$ [11].

Definition 3.9 (Differential Privacy).
A randomized mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{R}$ is $(\varepsilon,\ \delta)$-differentially private, with $\varepsilon > 0$ and $\delta \geqslant 0$, if for all $\mathcal{S} \subseteq \mathcal{R}$ and for all neighbouring datasets $\mathcal{D}$, $\mathcal{D}' \in \mathcal{D}$:

$$\Pr[\mathcal{M}(D) \in \mathcal{S}] \leqslant exp(\varepsilon) \Pr[\mathcal{M}(D') \in \mathcal{S}] + \delta$$

Important differential privacy mechanisms used throughout the thesis are described below (see also [11]).

Sensitivity To describe the maximum difference between two datasets due to the adding or removing of a single data point, the sensitivity is used. The worst-case difference a single data point can cause is considered to determine the sensitivity. Formally, the sensitivity is written as $\Delta = max_{\mathcal{D} \sim \mathcal{D}'}(|M(\mathcal{D}) - M(\mathcal{D}')|)$

Counting Queries To add noise correctly to something, counting queries are often used. Those count the number of elements that satisfy a predefined property in $\mathcal{D}$. For example, when considering a dataset $\mathcal{D}$ with n trajectories, a valid request would be: "In how many trajectories occur station L_1?". The output of this request is a counting query which can be noised using, e.g., Laplace noise.

Sequential Composition When using a ε_1-DP mechanism that follows a ε_2-DP mechanism (one the same data), then the output of this cascading action is $(\varepsilon_1 + \varepsilon_2)$-DP.

Parallel Composition When we partition a dataset $\mathcal{D}$ into n disjoint subsets and run an ε-DP mechanism on each of those n subsets, than releasing all n outputs is ε-DP

Post-Processing Theorem Any differential private mechanism M output is immune to post-processing. More formally, any individual cannot generate a function that uses the (ε, δ) differential private model $M(\mathcal{D})$ that has a less differential private model $M(\mathcal{D}')$ as output.

3.2.1 Laplace Mechansim

The Laplace Mechanism $\mathcal{M}_{L,q,\varepsilon}$ [12] is ε-DP and takes a database $\mathcal{D}$ as input and outputs a noised Δ_q-sensitivity bounded query q: $\mathcal{M}_{L,q,\varepsilon}(D) \mapsto q(D) + \text{Lap}(0, \frac{\Delta_q}{\varepsilon})$. The sensitivity Δ_q describes how much the output could change in the worst-case if a single data point is exchanged and $\text{Lap}(\Delta_q/\varepsilon)$ is defined as $\text{Lap}(\mu, \Delta_q/\varepsilon)[o] := \frac{\varepsilon}{2\Delta_q}\exp(-|o-\mu|\varepsilon/\Delta_q)$. Thus, a smaller ε or a larger sensitivity Δ_q corresponds to noise with a larger standard deviation.

Theorem 3.10 (The Laplace Mechanism is DP).
For a Δ_q-bounded ($\Delta_q \in \mathbb{R}_+$) counting query q and an $\varepsilon > 0$, the Laplace Mechanism $\mathcal{M}_{L,q,\varepsilon}$ is ε-DP.

Figure 3.4 shows an example of the Laplace noise distribution. As we can see, the more we get to the mean of the distribution, the more noise is added to the queries.

3.2.2 Rejection Sampling

Rejection sampling in Algorithm 1 is a technique to privately select candidates from a mechanism $M(\mathcal{D})$ that outputs x (e.g., a Trie) and a score q (e.g., a noisy F1-Score). It is ε-DP with $\varepsilon = 2 \cdot \varepsilon_1 + \varepsilon_0$ if M is ε_1-DP [17, Algorithm 1]. With a pre-defined threshold τ, we accept and return (x, q) if $q \geqslant \tau$, otherwise we repeat the process at most T rounds until either the output is accepted or a γ biased coin is true.

Theorem 3.11 (Rejection Sampling is $2\varepsilon_1 + \varepsilon_0$-DP).
For any $\varepsilon_0, \gamma > 0$, and an intger $T \geqslant \max\{\frac{1}{\gamma}\ln(\frac{2}{\varepsilon_0}), 1 + \frac{1}{e\gamma}\}$, if M is ε_1-DP, than the output of the rejection sampling is $2\varepsilon_1 + \varepsilon_0$-DP.

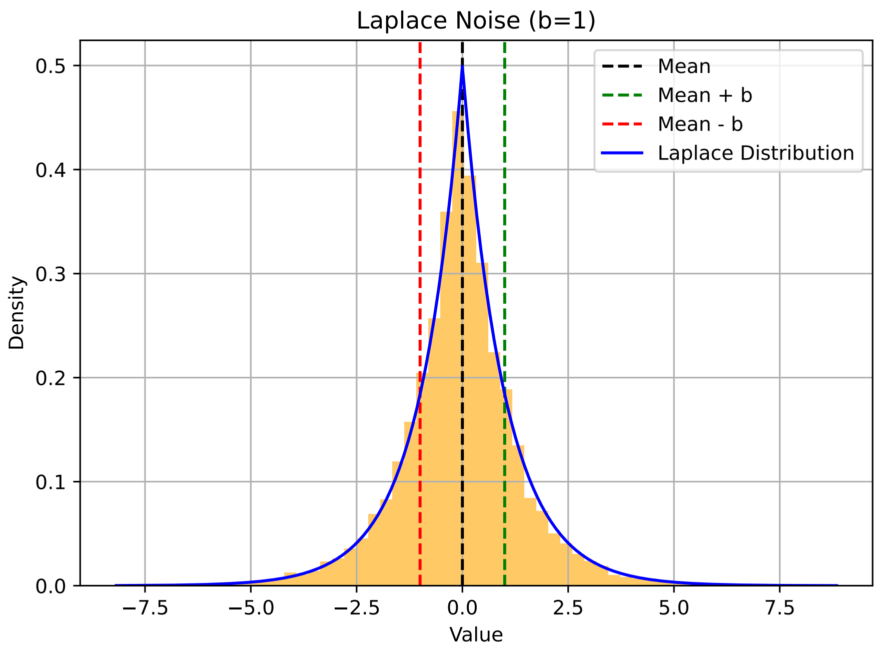


Figure 3.4: Example figure for the Laplace noise distribution. The closer one gets to the mean of the distribution, the more noise is added.

Algorithm 1 DP Rejection Sampling [17, Algorithm 1]

```
   **Input:** threshold $\tau$, probability $\gamma \leqslant 1$, privacy budget $\varepsilon_0 \leqslant 1$, number of steps $T \geqslant \max\left(\frac{1}{\gamma}\ln\frac{2}{\varepsilon_0}, 1+\frac{1}{e\gamma}\right)$, $\varepsilon_1$-DP mechanism $M(D)$
1: **for** $j = 1, \ldots, T$ **do**
2:     draw $(x, q) \sim M(\mathcal{D})$
3:     **if** $q \geqslant \tau$ **then**
4:         **return** $(x, q)$
5:     **end if**
6:     flip $\gamma$-biased coin s.t. with probability $\gamma$: **return** $\perp$
7: **end for**
8: **return** $\perp$
```

4

Problem Statement

This chapter describes the motivation of this thesis by going over the issue of releasing datasets containing movement patterns and explaining problems with current work. We will then have a look at some challenges we have faced and what threat models we consider.

4.1 Motivation

The analysis of mobility data is becoming increasingly relevant. In modern smart-city systems, operators record and analyse (often by a third party) the movement patterns of users of public transportation systems. However, the movement of a single user poses a risk of re-identification, as many users commute along the identical origin-destination pairs daily. Those patterns allow adversaries to learn about an individual's home and workplace. Thus, releasing this data without sanitising violates the privacy of individuals, making mechanisms to protect them relevant to enable the analysis of this data. Figure 4.1 demonstrates the problem with publishing original mobility patterns of public transport systems. The stars in the figure show the user's end stops. The more red a path appears the more users have taken this path. In such paths, individuals hide in the mass, but fewer users travel there when looking at the star at the far right. Making it easier for an adversary to learn about the individuals visiting this place frequently, making targeted surveillance (or stalking) easier.

Even if names and all other identifiers are stripped before releasing the data, the $A \rightarrow B$ commute may appear hundreds of times, hiding individuals in the mass. On the other hand, a trip $C \rightarrow D$ may occur only twice, making those individuals easily re-identifiable. As recently shown by CCC (Chaos Computer Club), the risk of re-identification based solely on movement patterns is more relevant than ever [3].

4.1.1 Limits of Existing Data and Methods

As publishing such mobility data poses numerous risks, there are no publicly available datasets. Even though datasets show sequential patterns, e.g., MSNBC [15], the goal of this thesis is to enable the publication of mobility data. Mobility sequences differ from other sequential data in two key ways:

1. Each user contributes at least two stops. Publicly available sequential pattern datasets

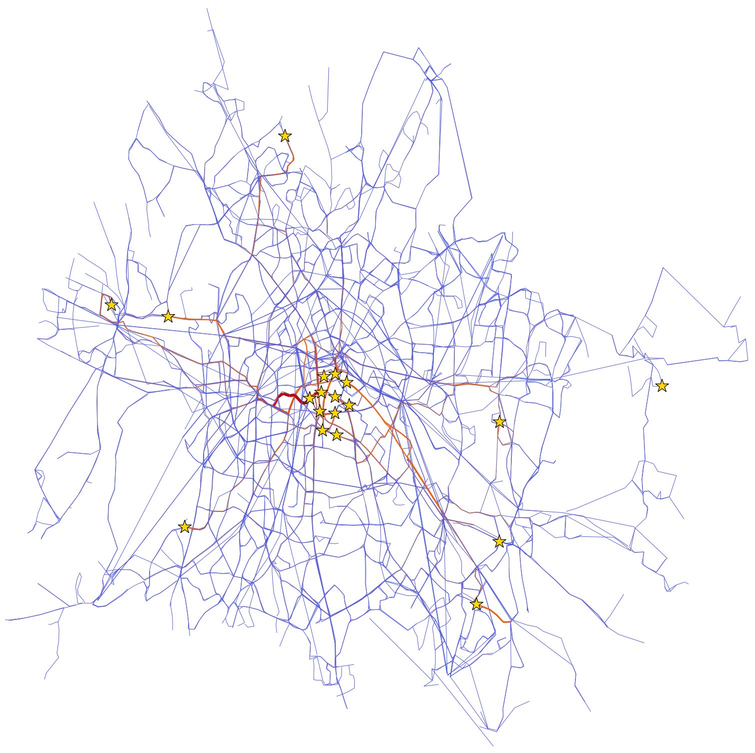


Figure 4.1: A heat-mapped trajectory graph of Berlin's transit network. Star markers show Endpoints. The path colour gets red the more users take this path.

often have trajectories containing only a single data point.

2. A user is never able to contribute the same stop directly after each other, meaning $[L_1, L_1]$ is never possible. However, such looping behaviour is often spotted in publicly available datasets, such as in the MSNBC dataset [15].

Nonetheless, publishing datasets containing trajectories (Definition 3.1 on page 9) is often desired. Primarily for research purposes, to prove the correctness of implementations. Companies maintaining these transport systems are also interested in analysis, as it reveals the necessity for expanding or cancelling specific routes with more trains, buses, etc. Since simple publishing of those datasets is impossible, techniques exist to create privacy-preserving prefix tree structures on mobility datasets [6, 7]. The downside of these approaches is that while the trees provide sufficient privacy guarantees, the trees construct paths that are not possible, resulting in a utility loss. Chen et al. [7] build pure prefix trees. The more location symbols are introduced, the more depth the tree gains, resulting in a state explosion and increasing noise levels at deeper tree nodes. Those problems are addressed by Chen et al. [6] by utilising n-grams. By building n-gram prefix trees, the height of the trees is controlled, decreasing the noise levels significantly. However, the building process brings every possible prefix to each level of the tree, constructing paths that never occurred and, therefore, are potentially not possible in the real world.

Furthermore, both approaches only consider the dataset given as the known universe of n-grams. This assumption holds in their proofs but is not applicable in the real world. As datasets are often a subset of the given transport network, some stops are not included. An adversary can infer this difference by considering two neighbouring datasets $D \sim D'$. Suppose D' includes a stop not present in D. In that case, Chen et al.'s [6] mechanism is unaware of it, violating differential privacy, as the adversary can infer whether that stop was present in D or not.

4.1.2 Our Approach

The problems with publicly available sequential datasets lead us to the implementation of a simulation scheme mimicking the movement of users on real-world transport systems. For this, we take public GTFS data to create a transport network. By using the location information of the stops and which lines drive from which stop, we can build a realistic network, simulating hotspot-based origin and destination movements (Chapter 5).

To address the utility loss of pure noise addition and the lack of real-world application, we construct a differentially private n-gram prefix tree over trajectories. This structure lets us

1. Publish noised counts at each prefix, using the Laplace mechanism
2. Generate synthetic trajectories by sampling paths
3. Reject poorly fitting trees via an F1-based rejection sampling loop [17]

The implementation discussed later on (Chapter 6 on page 31) only considers 3-grams, as these provide a good overview of transfers. To mitigate the problem of stops not being present in the provided dataset, we Additionally consider the complete universe of 3-grams possible in the underlying transport network. This universe is merged with the 3-grams provided by the dataset so that the counts match the weights of the data. By utilising the Laplace mechanism [12] to noise the count of 3-grams and selecting all significant 3-grams based on a random threshold between 0 and the standard deviation of Laplace noise, the prefix trees generated give an overview of the most significant transfers. After one tree is built, rejection sampling [17] is used to check the quality of the tree. For this, we check the F1 score, whenever the tree has a score over a threshold τ, it is accepted. Otherwise, a new tree is generated.

4.2 Challenges

This section discusses the challenges considered throughout the thesis. Those challenges are mainly meeting certain privacy guarantees, that the generated n-gram prefix trees are precise, and that the utility of the data is preserved.

Simulation As users follow patterns and some stops are more frequently used than others, we can not just randomly select where each user starts and ends their trip. This leads us to the implementation of a hotspot selection, ensuring that the simulated users are more likely to end their trip at one of the $15 - 30$ hotspots rather than a random end stop.

Privacy The algorithm must generate privacy-preserving prefix trees. Differentially private mechanisms generate a certain amount of noise depending on ε whenever the ground truth data is accessed and used. This ensures that even though the data is used for the computation, no information is leaked, thus not violating the privacy of individuals.

Precision While ensuring privacy is one of the main goals, it is also important to ensure that all trees returned are precise. This means that the infrastructure of the public transport system must align with the underlying dataset. The trees must not create paths between stops, which is not possible. Such edges would immediately reveal noise artefacts, so they cannot be published or relied upon.

Utility As all returned trees should be reasonably precise and private, the information gathered from the resulting trees should also be accurate. For this, we must have a certain degree of utility by preserving common trajectories by keeping the relative count of the frequencies.

4.3 Threat Models

In this section, two threat models are discussed. Those threat models are the main ones assumed for the resulting n-gram prefix trees. While we mainly assume the honest threat model, the malicious adversary must also be considered. For both, we briefly discuss what we assume exactly and how we protect against those.

We assume that the underlying dataset remains secret. Thus, the adversary has access only to the published prefix tree and to our DP algorithm on neighbouring datasets.

Honest For this threat model, we assume that the receiver of the prefix tree is honest. Therefore, it might be possible for him to gain information about the dataset by analysing the prefix tree. Such information gathering must always be considered, as this is part of the basic analysis of models. Since we use differential privacy, the algorithm proposed is ε-DP (Lemma 6.3), which restricts all inferences from the honest adversary.

Malicious In this scenario, the prefix tree receiver wants to gain information about the underlying dataset actively. As we use a tree structure for the model, a malicious adversary can search the tree, which enables a Membership Inference attack. By accessing the model, the adversary can check whether a trajectory or part of a trajectory (n-gram) was present in the dataset. Therefore, we assume the adversary can access the implementation and generate new outputs using two neighbouring datasets. However, since the implementation is ε-differentially private (Lemma 6.3), it is considered sufficient protection against such attacks, as we know from [5, 19].

5

Simulating Mobility Data

This chapter examines the implementation of a public transport system simulation. For this, we look at the idea behind the simulation, how we ensure that not only uniformly random data is generated, and what techniques are used to compute the trajectories.

5.1 Idea

Most publicly available sequential pattern datasets are not suited for evaluating mobility data processing algorithms. As those datasets show behaviours not possible in mobility patterns, such as:

1. Each user contributes at least two stops. Publicly available sequential pattern datasets often have trajectories containing only a single data point.
2. A user is never able to contribute the same stop directly after each other, meaning $[L_1, L_1]$ is never possible. However, such looping behaviour is often spotted in publicly available datasets, such as in the MSNBC dataset [15].

Due to this, this chapter aims to create a simulation tool that allows the reading of publicly available GTFS data and generates a mobility simulation. In this simulation, the number of users generating trajectories can be adjusted, allowing for control over the data space generated. For the simulation to work as intended, the following steps are made:

1. Initially, the GTFS data is filtered and processed so that trajectories can be constructed using the A* algorithm.
2. After that, we sample hotspots so that the generated trajectories have variance and are not just uniformly random generated.
3. Now, to construct trajectories, we use the A* algorithm to find the best path from one station to another. All stations taken during that are added to the trajectory of the current user.
4. We use the Haversine distance as the heuristic function for the A* algorithm to compute the least number of stops needed to find the best path of transport stops.

5.2 Pre-Process Data

GTFS data is a collection of files that contain records of the public transport system of a city, state, or country. The most relevant files for us are the ones containing the stop locations and the routes of each available line.

Those files are needed, as GTFS data is usually not available for a city. Therefore, it is necessary to filter out all stops that are not part of the city's transport system. This can be done by checking the coordinates of the stops and verifying if they are located within a specific area.

Algorithm 2 Filter: Filters the stops

```
Require:
    File containing stops $f$
    list of boundaries $bound$
1: $stopsToUse$ ← [ ]                              ▷ Initialize empty list
2: for $stop$ in f do
3:     if coordiantes of $stop$ in $bound$ then
4:         $stopsToUse$.append(stop)               ▷ Append stop to list
5:     end if
6: end for
7: return $stopsToUse$
```

Algorithm 2 gets the file of all stops f and the list of boundaries *bound* for which we accept a stop from *f*. Doing so generates a list that only contains all stops within the given boundaries, which are then used in further parts of the simulation to compute trajectories based on, e.g. a city. For that, the algorithm iterates over all stops in f. If the coordinates of the current stop are within *bound*, the stop is added to the list of usable stops *stopsToUse* (lines 2-6). Ultimately, the list of all accepted stops is returned (line 7).

5.3 Hotspot Generation

To avoid having uniformly random data, the definition of hotspots is essential. It creates a sink that ensures, with a higher probability, that the simulated users end their trip at a hotspot. This gives control over the simulation's behaviour and avoids having uniformly random data with no logical patterns.

Algorithm 3 takes the list of acceptable stops *stopsToUse* (Algorithm 2) to generate 15 to 30 hotspots, which are given a higher probability of being selected. By iterating over a random number between 15 and 30, the algorithm checks if the stop at some random index of the *stopsToUse* list has already been selected. If yes, a while loop is used to find a new index (lines 9-11). If the index has not been selected yet, it is appended to *hotspots* (lines 5-8). Finishing the selection process (lines 4-14), the weights of all stops are generated (lines 15-22). For this, we iterate over all stops in *stopsToUse*. If the current stop is a hotspot, its weight is computed ($len(stopsToUse) - len(hotspots) + 1.0$). This weight is then appended to the *probabilities* list (lines 16-18). Otherwise, 1.0, the base weight, is appended to *probabilities* (line 20). This weighting is done as not all users of

Algorithm 3 Hotspots: Hotspot generation

```
Require:
    filtered stops $stopsToUse$                                  ▷ Algorithm 3
 1: $hotspots$ ←[ ]                                   ▷ Initialize empty hotspot list
 2: $probabilities$ ←[ ]                       ▷ Initialize empty list of probabilities
 3: baseWeight ←1.0                                           ▷ Define base weight
 4: for i in randomInt(15, 30) do
 5:     i* ←$stopsToUse$[randomInt(1, len($stopsToUse$))]        ▷ Get random index
 6:     if $stopsToUse[i^*]$ not in hotspots then
 7:         $hotspots$.append($stopsToUse$[i*])              ▷ Append item at that index
 8:     else
 9:         while $stopsToUse$[i*] in hotspots do                ▷ Search new index
10:             i* ←$stopsToUse$[randomInt(1, len($stopsToUse$))]    ▷ Get random index
11:         end while
12:         $hotspots$.append($stopsToUse$[i*])              ▷ Append item at that index
13:     end if
14: end for
15: for stop in $stopsToUse$ do                               ▷ Generate probabilities
16:     if stop in $hotspots$ then
17:         weight ←$len(stopsToUse) - len(hotspots)$ + baseWeight
18:         $probabilities$.append(weight)        ▷ Hotspots have higher probabilities
19:     else
20:         $probabilities$.append(baseWeight)
21:     end if
22: end for
23: return [($probabilities$ / sum($probabilities$))]         ▷ normalize all weights
```

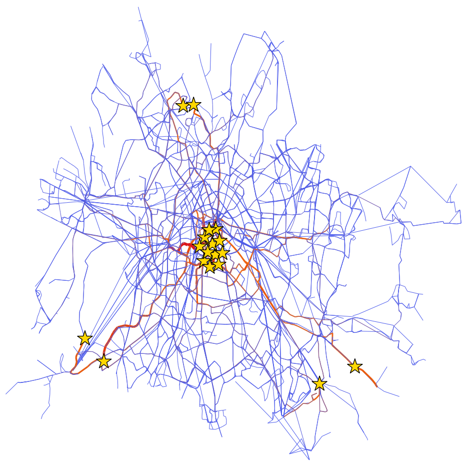
public transport systems drive to distinct hotspots. Finally, the list of normalized weights is returned (line 23).

Figure 5.1 shows the difference between a uniformly random and biased simulation. As we can see, it is more realistic when adding biased sinks since most users of public transport systems have specific locations they travel to (e.g., work, home, points of interest).

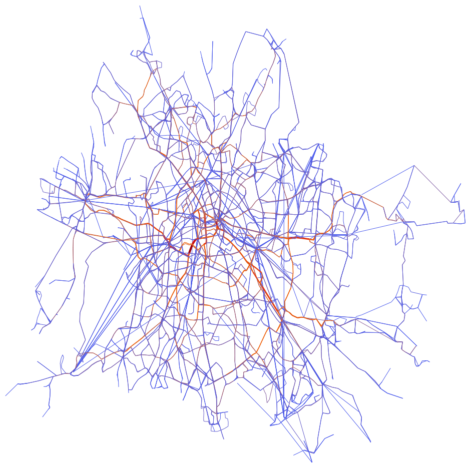
5.4 Haversine Distance

The Haversine distance d is a formula for computing the great-circle distance between two points on the surface of a sphere, given their latitudes (ϕ) and longitudes (λ). Unlike planar approximations of distances, it accounts for the Earth's curvature, making it especially suitable for geospatial analysis where accuracy over long distances is required. The Haversine distance uses the Haversine function, defined as:

$$hav(\Theta) = \sin^2\left(\frac{\Theta}{2}\right) = \frac{1 - \cos(\Theta)}{2}$$



(a) Simulation using the hotspot generation. Stars mark the position of a hotspot.



(b) Simulation without the hotspot generation.

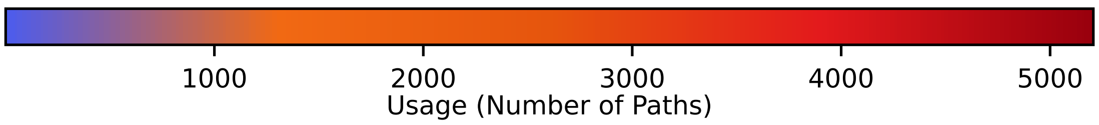


Figure 5.1: Differnence between a completely random simulation (Figure 5.1b) and when using weighted randomness through the hotspot generation (Figure 5.1a). For better readability of those figures see Section A.2.

With Θ denoting the central angle between two points, measured from the centre of a sphere. This means when drawing lines from the centre of a sphere to those points, the angle between those is Θ. Meaning the actual surface distance d is related to Θ by

$$d = r \cdot \Theta$$

Showing over the spherical law of cosines:

$$\cos = \sin(\phi_1) \cdot \sin(\phi_2) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\lambda_2 - \lambda_1).$$

By now recasting this term using the Haversine formula, we get the Haversine distance d as follows.

$$
\begin{aligned}
\operatorname{hav}(\Theta) &= \sin^2\left(\frac{\phi_2-\phi_1}{2}\right) + \cos(\phi_1)\cdot\cos(\phi_2)\cdot\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)\\
\Rightarrow \Theta &= 2\arcsin\left(\sqrt{\operatorname{hav}(\Theta)}\right)\\
\Rightarrow d &= 2r\cdot\arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right) + \cos(\phi_1)\cdot\cos(\phi_2)\cdot\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)\\
&= 2r\cdot\arcsin\left(\sqrt{\frac{1-\cos(\phi_2-\phi_1)+\cos(\phi_1)\cdot\cos(\phi_2)\cdot(1-\cos(\lambda_2-\lambda_1))}{2}}\right)
\end{aligned}
$$

The following holds:

- r is the radius of the sphere (e.g., Earth's radius),
- ϕ_1, ϕ_2 are the latitudes of two points in radians,
- λ_1, λ_2 are the longitudes of two points in radians.

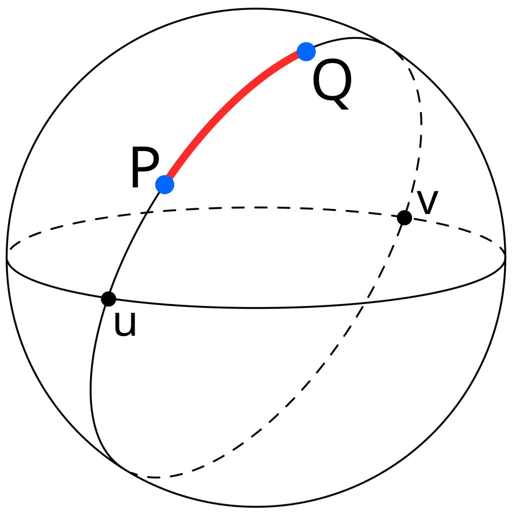


Figure 5.2: Illustration of the Great-Circle distance of two points (P and Q) on a sphere [24]. The points u and v are antipodal points.

5.5 A* Algorithm

Given a source and a sink, the A* algorithm traverses some graphs and finds the shortest path between those two nodes. The optimal path is found by estimating the cost of traversing the current path. By doing so for all paths, the path with minimal cost is found. Estimating the cost to reach the goal node uses a heuristic function. For the simulation, we use the Haversine distance (Section 5.4).

Algorithm 4 shows the implementation of the A* algorithm. Initially, no paths have been explored yet, so we set both the actual cost estimated $g[v]$ and the total projected

Algorithm 4 A*: A* Algorithm for Shortest Path

```
Require:
    Graph $G = (V, E)$
    start node $s$
    goal node $t$
 1: Initialize min-heap:   $openSet \leftarrow \{s, h(s,t)\}$            ▷ Nodes to be evaluated
 2: Initialize empty set:   $visitedSet \leftarrow \emptyset$          ▷ Tracks fully evaluated nodes
 3: Initialize empty map:   $cameFrom \leftarrow \{\}$   ▷ Stores optimal predecessor of each node
 4: For each $v \in V$: set $g[v] \leftarrow \infty$ and $f[v] \leftarrow \infty$                ▷ Cost estimates
 5: $g[s] \leftarrow 0$                                   ▷ Cost from start to itself is zero
 6: $f[s] \leftarrow h(s,t)$                          ▷ Heuristic estimate (Haversine distance)
 7: while $openSet \neq \emptyset$ do
 8:     $(u, \_) \leftarrow$ popMin($openSet$)                   ▷ Extract node with smallest $f$
 9:     if $u = t$ then
10:         return ReconstructPath($cameFrom, t$)                        ▷ Path found
11:     end if
12:     $visitedSet \leftarrow visitedSet \cup u$                           ▷ Update visited set
13:     for all neighbours $v$ of $u$ do
14:         if $v \in visitedSet$ then
15:             continue
16:         end if
17:         $tmpG \leftarrow g[u] + \mathrm{dist}(u, v)$                      ▷ Cost to reach $v$ from $u$
18:         if $tmpG < g[v]$ then
19:             $cameFrom[v] \leftarrow u$
20:             $g[v] \leftarrow tmpG$
21:             $f[v] \leftarrow g[v] + h(v,t)$             ▷ Update actual cost to $v$ with heuristic
22:             if $v \notin openSet$ then
23:                 $openSet \leftarrow openSet \cup (v, f[v])$                ▷ Update the min-heap
24:             else
25:                 decreaseKey($openSet$, $v$, $f[v]$)   ▷ Update priority of existing node $v$
26:             end if
27:         end if
28:     end for
29: end while
30: return failure                                ▷ No path exists between $s$ and $t$
```

cost estimated $f[v]$ to ∞, for every node v in the graph $G = (V, E)$ (line 4). This shows that the shortest distance from the start s to each node is unknown. We then explicitly assign $g[s] = 0$ since reaching the start node from itself has no cost. Furthermore, using the Haversine distance h as the heuristic function, we assign $f[s] = h(s,t)$ to determine the distance from s to t (lines 5-6). The while loop (lines 7-29) checks if a path exists. If not, the algorithm returns *failure*, indicating that no path can be found (line 30). At every while loop iteration, the node with the lowest cost is popped from the min-heap *openSet*, giving us u (line 8). Should u happen to be the goal node t, the algorithm returns the found optimal path (lines 9-11). Otherwise, we update *visitedSet* by adding u (line 12) and iterate over all its neighbours (lines 13-28). If the current neighbour v was already visited, we go on to the next neighbour of u (lines 14-16), but if v has not yet been visited, we check the cost of reaching v from u (lines 17-27). By adding up the current cost of u and the distance from u to v, we check if this temporary cost *tmpG* is less than the actual cost $g[v]$ (lines 17-18), if *tmpG* is greater than $g[v]$, we ignore v. However, if *tmpG* is less than $g[v]$, we update *cameFrom*, both $g[v]$ and $f[v]$, and update the min-heap *openSet*, either by adding v, or by updating the priority of v (lines 19-26).

Therefore, the A* algorithm either returns an optimal path from a given start s to a given goal node t, or no path is returned, making the algorithm complete as it guarantees to find a solution if one exists.

5.6 Generating the Simulation

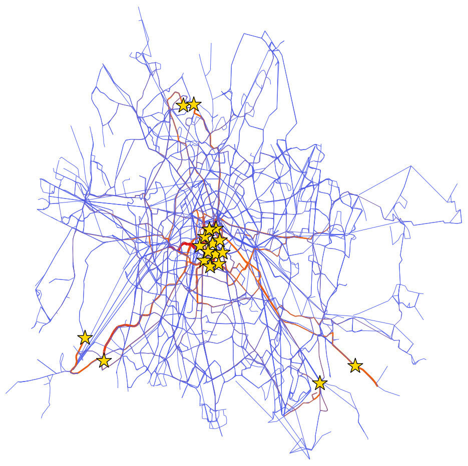
Now that the three main parts of the simulation have been introduced, we discuss how they are used to simulate the movement patterns of users in a public transportation system.

Algorithm 5 Simulation: Simulating movement patterns

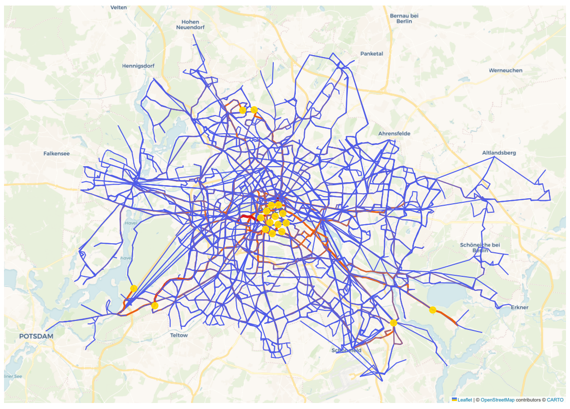
```
Require:
    File containing all stops $f$
    File containing all paths $c$
    list of boundaries $bound[x_{low}, x_{high}, y_{low}, y_{high}]$
    number of users to simulate $n$
 1: $stopsToUse$ ←Filter($f$, $bound$)                                   ▷ Algorithm 2
 2: $hotspots$ ←Hotspots($stopsToUse$)                                   ▷ Algorithm 3
 3: Graph $G$ ←$s$ as $V$ and $G$ ←$c$ as $E$                ▷ Construct $G$, using $s$ and $c$
 4: $trajectories$ ←[ ]                                          ▷ Initialize empty list
 5: for i in range($n$) do
 6:     startStop ←random($stopsToUse$)                       ▷ Get random stop from f
 7:     endStop ←random.choice($stopsToUse$, probability=$hotspots$)   ▷ Weighted random
 8:     path ←A*($G$, startStop, endStop)                                ▷ Algorithm 4
 9:     $trajectories$.append(path)                                 ▷ Append found path
10: end for
11: return $trajectories$                                ▷ Return list of all trajectories
```

Algorithm 5 takes a file including all stops f, a file that contains all connections of the transport system c, a list of the geographic boundaries (latitudes and longitudes) *bound*, and the number of users to simulate n, as input. Using f and *bound*, the usable stops are filtered by Algorithm 2 and saved in *stopsToUse* (line 1). This list is then used to generate the hotspot probabilities (line 2, Algorithm 3). We construct a graph G by assigning all stops in f as nodes of G, and all paths in c are the edges of G (line 3). After that, an empty list is initialized, which stores the computed trajectories from the A* algorithm (lines 4-10). By iterating over the number of users n, in each iteration, a random starting position from *stopToUse* is selected. The end stop is selected using weighted randomness based on the probabilities in *hotspots*, ensuring that with a higher probability, a hotspot is selected, rather than some of the other stops in *stopsToUse*. Finally, the selected starting- and ending stop alongside G is given to the A* algorithm to get the optimal path, which is appended to *trajectories* (lines 5-10). The algorithm finishes by returning the list of all found paths *trajectories* (line 11).

Figure A.4 shows an example of how the result of Algorithm 5 looks when plotting the trajectories over the map of the corresponding city, here Berlin. The stars mark the position of a hotspot. The blue lines indicate that a path is taken by a few users, the more red it gets, the more users take this path.



(a) Result of the simulation, showing the network and the hotspots.



(b) Result of the simulation with the corresponding map underneath the network.

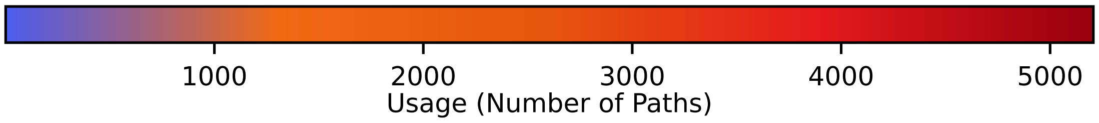


Figure 5.3: Simulation of 100,000 users taking the public transport system in Berlin, displayed as a heatmap. The more a path is taken, the larger and redder the connection is displayed. The trajectories are computed using the A* algorithm, with the Haversine distance as the heuristic function. Appendix A.2 shows the figures in full size.

6

MobiTrie (Mobility Prefix-Tree)

This chapter discusses the implementation of the Mobility Prefix-Tree (later called *MobiTrie*) algorithm. We first discuss the idea behind this algorithm. After that, the pre-processing of given trajectories, either from the simulation (Chapter 5) or the MSNBC dataset [15], is examined. With the pre-processed trajectories, we then build a 3-gram prefix tree (later on called trie, see Definition 3.8), which can be used for further analyses without violating the privacy of individuals, as *MobiTrie* is proven to fulfill differential privacy (Lemma 6.3).

6.1 Idea

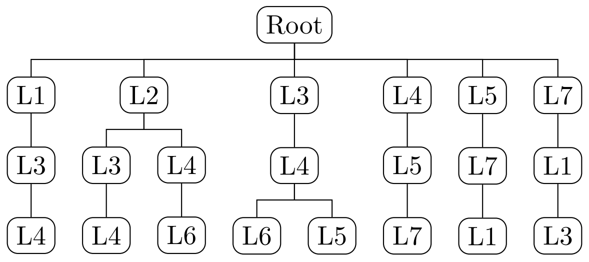
Graphical visualisations are a proven method for analysing mobility data, as they provide direct feedback on users' movement. However, those visualisations can not be published as they would reveal crucial information about users (see Chapter 4 on page 17). Therefore, the idea behind the *MobiTrie* algorithm is to make the publication of mobility data possible without revealing identifying information. For this reason, we performed the following steps to create the ε-DP *MobiTrie* algorithm:

1. At first, the given trajectories are used to create 3-grams. For each 3-gram, the frequency is counted. We consider 3-grams for the trie generation, as those give a good overview of transfers in a transport system.
2. The generated 3-grams are then used to generate a ε-DP trie, by utelizing rejection sampling (Section 3.2.2).

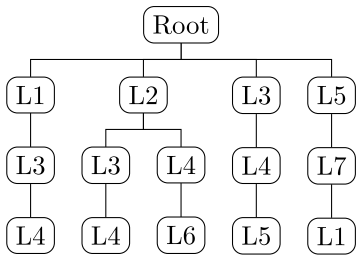
6.2 Pre-Processing Trajectories

Using the given trajectories from Chapter 5 and [15], we now compute 3-grams (see Definition 3.7). Mostly, such 3-grams are static, meaning that the first three locations of a trajectory are considered, and the rest of the trajectory is discarded, making the analysis and presentation of movement patterns more complicated. For *MobiTrie*, we use **rolling** 3-grams, as those ensure that each consecutive sequence of three locations in a trajectory is treated as a separate 3-gram. This ensures that the entire trajectory contributes to the analysis, providing a more detailed overview of movement patterns.

Looking at Table 3.1 and Figure 3.3b, we can see how such static 3-grams underrepresent the dataset. Reconsidering Table 3.1 a trie with rolling 3-grams looks like Figure 6.1a.



(a) Rolling 3-gram prefix tree.



(b) Static 3-gram prefix tree.

Figure 6.1: Trie with rolling- and static 3-grams, based on Table 3.1 on page 10. Those two tries represent the same data, with different 3-gram approaches.

As we can see by comparing the tries from Figure 6.1, the base structure is the same, but two branches are added, and the branch with $L3$ as a parent got an additional leaf. Those additional branches and leaves give us a better overview and utility of movement patterns, leading us to the following advantages of rolling 3-grams.

6.2.1 Advantages of Rolling 3-Grams

Rolling 3-grams provide several advantages compared to static 3-grams:

Comprehensive Analysis Rolling 3-grams capture more parts of a trajectory, as not only the first n locations contribute to the analyses.

Improved Data Representation By considering overlapping segments, rolling 3-grams provides a richer dataset representation, reducing the loss of information.

Enhanced Synthetic Data Quality The additional granularity in rolling 3-grams improves the quality of synthetic data generated from the trie, making it more representative of the original dataset.

These advantages make rolling 3-grams a powerful tool for mobility pattern analysis, especially in privacy-preserving contexts. Rolling n-grams are defined as follows.

Definition 6.1 (Rolling n-Grams).
Given a trajectory $\mathcal{T} = [L_1, L_2, \ldots, L_{|\mathcal{T}|}]$, as in Definition 3.1. For any integer n with $1 \leqslant n < |\mathcal{T}|$, a rolling n-gram $\mathcal{R}_n$ over $\mathcal{T}$ is defined as:

$$\mathcal{R}_n(\mathcal{T}) = \big((L_1, \ldots, L_n), (L_2, \ldots, L_{n+1}), \ldots, (L_{|\mathcal{T}|-(n+1)}, \ldots, L_{|\mathcal{T}|})\big).$$

Meaning equivalently, for a sliding window of size i:

$$\mathcal{R}_n(\mathcal{T}) = \big(\underbrace{(L_i, L_{i+1}, \ldots, L_{i+(n-1)})}_{i\text{th window}} : i = 1, \ldots, |\mathcal{T}| - n + 1\big).$$

Thus, rolling n-grams are constructed by sliding a window of size n over $\mathcal{T}$, capturing $|\mathcal{T}| - n + 1$ sequences in total.

As shown in Figure 6.1a with rolling 3-grams, we get all possibilities from a given dataset $\mathcal{D}$, enabling more precise analysis of movement patterns. This brings us to the following algorithm for a rolling 3-gram construction over a given trajectory dataset $\mathcal{D}$.

6.2.2 Constructing Rolling 3-Grams

Algorithm 6 gets as input some trajectory dataset $\mathcal{D}$, for us either from Algorithm 5 or [15]. Using $\mathcal{D}$, an empty list *result* is initialised (line 1) in which all constructed rolling 3-grams (Definition 6.1) are stored. By looping over all trajectories in $\mathcal{D}$ (lines 2-12), every iteration initializes an empty set *3GramsSet*, to avoid duplicates being stored (line 3). This ensures that a trajectory can contribute to the same 3-gram only once, giving us further guarantees. After checking if the current trajectory's length is sufficient for a 3-gram construction and that no more than 20 3-grams can be contributed (line 4), we loop over all locations in the current trajectory (lines 5-7). Each found 3-gram in the trajectory is added to *3GramsSet* (line 6). Should the current trajectory be able to contribute more than 20 3-grams, we first construct all 3-grams possible (lines 10-13) and then 20 random 3-grams are selected, which are appended to *3GramsSet* (line 14). After iterating over the whole trajectory, all found 3-grams are appended *result* (lines 16-18). Here, we want to have duplicates to count the frequencies of the 3-grams later on. Finally, the algorithm returns the list of all found 3-grams *result* (line 20).

6.3 Processing 3-Grams

To generate a trie, the 3-grams used must be processed accordingly. For that, the frequencies of all 3-grams must be counted, and we need a mechanism to select only those 3-grams considered significant. For those tasks, the following two algorithms are used.

Algorithm 7 counts the number of occurrences of all 3-grams in *result*. A dictionary is initialised for this, where all 3-grams in *result* get an initial count of 0 (line 1). After that we iterate over all 3-grams in *result* and increase the count of the current 3-gram according to its number of occurrences in *result* (line 3).

Algorithm 8 is used to filter the noised 3-gram frequencies by their significances. For this, the algorithm takes the dictionary of (3-gram, noisy count) pairs, *3Grams*, and the privacy budget used, ε, as input values. In the first step, an empty list *significant* is initialised, storing the tuples of (3-gram, noisy count) to be considered (line 1). After that a random value between 0 and $\frac{k\cdot\sqrt{2}}{\varepsilon}$ (the standard deviation of Laplace noise) is selected as the threshold τ (line 2). By now, iterating over all 3-grams in *3Grams*, it is checked if the noisy count of the current 3-gram is greater than or equal to τ, if yes, (3-gram, noisy count) is appended to *significant* (lines 3-7). In the end, the algorithm returns *significant*, which only has all (3-gram, noisy count) tuples considered significant (line 8).

Algorithm 6 3Gram: Rolling 3-Gram construction

```
Require:
    trajectory dataset $\mathcal{D}$
 1: $result$ ←[ ]                                          ▷ Initialize empty set
 2: for trajectory in $\mathcal{D}$ do
 3:     $3GramsSet$ ←∅                                     ▷ Initialize empty set
    ▷ Check if 3-gram is constructible and only 20 3-grams are contributed
 4:     if len(trajectory) $\geqslant$ 3 and len(trajectory) $\leqslant$ 20-2 then
 5:         for i in range(len(trajectory) -2) do
 6:             $3GramsSet$ ←$3GramsSet$ ∪ {(trajectory[i], trajectory[i+1], trajectory[i+2])}
 7:         end for
 8:     end if
    ▷ If more than 20 3-grams can be contributed, select 20 random 3-grams
 9:     if len(trajectory) $> 20 - 2$ then
10:         $All3Grams$ ← ∅                                ▷ Collect all possible 3-grams
11:         for i in range(len(trajectory) -2) do
12:             $All3Grams$ ← $All3Grams$ ∪ {(trajectory[i], trajectory[i+1], trajectory[i+2])}
13:         end for
14:         $3GramsSet$ ← RandomSample($All3Grams$, 20)     ▷ Pick 20 random 3-grams
15:     end if
16:     for all 3-Grams in $3GramsSet$ do
17:         $result$.append(3-Grams)                       ▷ Append all found 3-Grams
18:     end for
19: end for
20: return $result$                                        ▷ Return set of all 3-grams
```

6.4 Contructing the Trie

As we previously introduced the pre-processing of trajectories and the counting and selection process of 3-grams, we now discuss how these algorithms are used to create a differentially private trie. Before that, we need to discuss how quadruple of (TP, FP, FN, TN) of the trie is built (Algorithm 9), as the true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) are needed to compute the F1 score in the rejection sampling process.

Algorithm 9 needs the set of (3-gram, count) tuples $\mathcal{G}$ given in the trajectory dataset $\mathcal{D}$, the set of all (3-gram, noisy count) tuples used to build the trie $\mathcal{C}$, and the universe of all possible 3-grams in the transport network $\mathcal{U}$ as input. Then, we initialise the quadruple (TP, FP, FN, TN), with counts all set to 0 initially (line 1). By iterating over the universe $\mathcal{U}$, we check whether the 3-gram g is in G or C (lines 2-15). Those checks correspond to the definition of TP, FP, FN, and TN, which is described as follows for some 3-gram g.

TP If $t \in G$ and $g \in C$
FP If $t \notin G$ and $g \in C$
FN If $t \in G$ and $g \notin C$
TN If $t \notin G$ and $g \notin C$

Algorithm 7 Counting: Get the frequency of all 3-Grams

```
Require:
    list of 3-grams *result*
1: $counts \leftarrow \{n : 0\}\ \{\forall n \in\ result\}$    ▻ Initialize dictionary of all 3-grams
2: for 3Gram in *result* do
3:     *counts*[3Gram] ←*counts*[3Gram] +1    ▻ Increment count in dictionary
4: end for
5: return *counts*
```

Algorithm 8 Select: Select the most significant 3-grams

```
Require:
    dictionary of (3-gram: noisy count), pairs *3Grams*
    privacy budget $\varepsilon$
1: *significant* ←[ ]    ▻ Initialize empty list
2: $\tau \leftarrow$random(0.0, $\frac{\sqrt{2}\cdot 20}{\varepsilon}$)    ▻ Select random threshold $\tau$ between 0.0 and $\frac{\sqrt{2}}{\varepsilon}$
3: for (n, cnt) in *3Grams* do
4:     if cnt $\geqslant \tau$ then
5:         *significant*.append((n, cnt))    ▻ Only keep significant 3-grams
6:     end if
7: end for
8: return *significant*
```

For the case where t fulfills one of those conditions, the corresponding value is incremented by the count given from t (lines 6-14). Ultimately, the algorithm returns the computed quadruple (line 16).

Algorithm 10 brings Algorithm 6, 7, 8 and 9 together, and by utilizing rejection sampling (Section 3.2.2) and Laplace noise (Section 3.2.1) a ε-DP trie is constructed. For the algorithm to work the universe of all possible 3-grams $\mathcal{U}$ of the underlying graph, some finite trajectory dataset D, some threshold value τ, a probability γ, privacy budgets ε and ε_0, and two budget shares $r_1$and r_2 are required as input values. At the beginning, the number of rejection sampling steps T is computed using γ and ε_0 (line 1). Using D, we construct and count all 3-grams by utilising Algorithms 6 and 7 (lines 2-3). After that, we need to initialise our ε values. This step is necessary as we want to achieve ε differential privacy, therefore, the input ε must be split among all mechanisms using it. We use our shares r_1 and r_2, with $r_1 + r_2 = 1$ to create such splits. By performing $\varepsilon_1 = 0.5 \cdot (\varepsilon - \varepsilon_0)$ and than splitting ε_1 into $\varepsilon_{cnt} = r_1 \cdot \varepsilon_1$ and $\varepsilon_{f1} = r_2 \cdot \varepsilon_1$ (lines 4-6), we get $\varepsilon = 2 \cdot \varepsilon_1 + \varepsilon_0$ givening us our ε-DP guarantee (see Lemma 6.3). As we need to minimise the influence of unseen stops, the universe of all 3-grams possible in the underlying transport system $\mathcal{U}$ is used. Thus, we create a union of *3Gram* and $\mathcal{U}$, so that all 3-grams not present in *3Gram* get a base score of 1.0 (lines 7-11). Such 3-grams exist because $\mathcal{D}$ may not cover all possible paths of the transport system. To compensate for this, we need $\mathcal{U}$, this also ensures that when giving a dataset of the same transport system but with different movement patterns, the tries generated essentially are not completely different than before. This also guarantees that adversaries gain less information when using neighbouring datasets. In each iteration

Algorithm 9 Quadruple: Compute (TP, FP, FN, TN) for a trie-based 3-gram classifier

```
Require:
    Set of 3-grams $\mathcal{G}$                ▷ Set of all true (3-gram, count) tuples (ground truth)
    Set of trie 3-grams $\mathcal{C}$  ▷ Set of all (3-gram, noised count) tuples recognized by the trie
    Universe of all possible 3-grams $\mathcal{U}$        ▷ Set of all 3-grams of the transport network

 1: $(TP, FP, FN, TN) \leftarrow 0, 0, 0, 0$                    ▷ Initialize quadruple values
 2: for all $g \in \mathcal{U}$ do
 3:     $in\_G \leftarrow (g \in \mathcal{G})$
 4:     $in\_C \leftarrow (g \in \mathcal{C})$
 5:     if $in\_G \wedge in\_C$ then
 6:         $TP \leftarrow TP++$
 7:     else if $\neg in\_G \wedge in\_C$ then
 8:         $FP \leftarrow FP++$
 9:     else if $in\_G \wedge \neg in\_C$ then
10:         $FN \leftarrow FN++$
11:     else
12:         $TN \leftarrow TN++t$
13:     end if
14: end for
15: return $(TP, FP, FN, TN)$                              ▷ Return quadruple of all values
```

of the rejection sampling (Section 3.2.2 on page 14), we create a copy of *3Gram* so that we are not noising already noised values (line 13). All 3-gram count tuples of the copy are noised using Laplace noise (Section 3.2.1) (lines 14-16). Finishing the noising step, the list of (3-gram, noisy count) tuples alongside the used ε_{cnt} are given to Algorithm 8 so that we get a filtered list *DP-3Gram* of (3-gram, noisy count) tuples, we want to build a trie with (lines 17-18). Based on the built trie and the trajectory dataset D, Algorithm 9 builds the quadruple of TP, FP, FN, and TN (line 19). In the next step, we noise all four values of the quadruple, using Laplace noise with a scale of $\frac{40}{\varepsilon_{f1}}$ (line 20). Using those noisy values, we compute $F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$, which we use to estimate the utility of the trie (line 21). If $F1$ is greater than or equal to the given threshold τ, we accept the trie and return it (lines 22-24). Otherwise, a γ biased coin is flipped, and with probability γ, we halt and output $\perp$ (line 25). Should in all T iterations never be a trie accepted, and the γ biased coin is also never true. The algorithm outputs $\perp$, as no trie could be found (line 27).

6.4.1 Possible Tries

The following shows possible resulting tries using different ε values. Additionally, the base trie generated without rejection sampling and noise is shown to enable comparison with the ε-DP tries.

All tries seen here can be found in Appendix B in more detail.

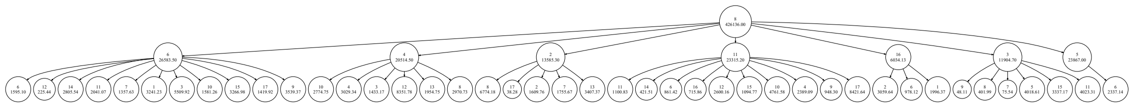


Figure 6.2: Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.1$.

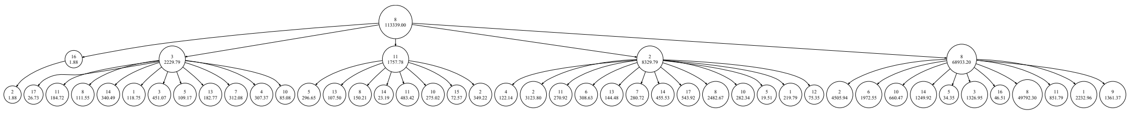


Figure 6.3: Possible trie of the MSNBC dataset [15], using $\varepsilon = 1.0$.

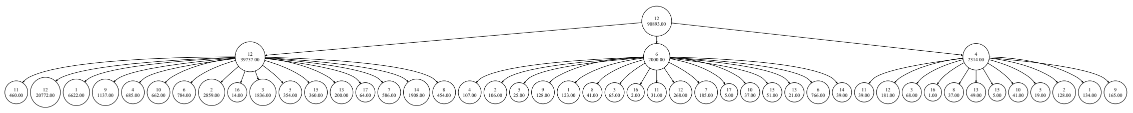


Figure 6.4: Baseline trie of the MSNBC dataset [15]. The baseline indicates that no differential privacy mechanisms were employed in its creation.

6.4.2 Privacy Guarantee

MobiTrie (Algorithm 10) is ε-differentially private, which is proven below. Before proving that our privacy guarantees hold, we prove that the quadruple (TP, FP, FN, TN) has sensitivity $\Delta_{quad} = k$, for a finite number of trajectories and 3-grams, and a maximum of 20 3-grams contributed per trajectory.

Lemma 6.2 (Quadruple (TP, FP, FN, TN) Sensitivity).
Let $\mathcal{D}$ be a dataset of trajectories, each trajectory contributing at most 20 distinct 3-grams. Fix an integer threshold $\tau > 0$. Define $\mathcal{U}$ as the universe of all possible 3-grams, in a transport network. Define the ground truth set $G(\mathcal{D})$ of 3-gram, count pairs as:

$$G(\mathcal{D}) = \{g : g \textit{ is a 3-gram appearing in } \mathcal{D} : \mathcal{D} \subseteq \mathcal{U}\},$$

Then for an arbitary prediction set $S \subseteq \mathcal{U}$, define the binary label for each $g \in S$ as positive if $g \in G(\mathcal{D})$ and negative otherwise.

Then for all $\mathcal{D}$ one can compute the quadruple counts:

$$\begin{aligned}
\mathrm{TP}(S, G(\mathcal{D}')) &= \big|\{g \in S : g \in G(\mathcal{D}')\}\big|,\\
\mathrm{FP}(S, G(\mathcal{D}')) &= \big|\{g \in S : g \notin G(\mathcal{D}')\}\big|,\\
\mathrm{FN}(S, G(\mathcal{D}')) &= \big|\{g \notin S : g \in G(\mathcal{D}')\}\big|,\\
\mathrm{TN}(S, G(\mathcal{D}')) &= \big|\{g \notin S : g \notin G(\mathcal{D}')\}\big|.
\end{aligned}$$

Define the Δ_{quad} sensitivity of the quadruple (TP, FP, FN, TN) *as*

$$\Delta_{quad} = \max_{\mathcal{D}\sim\mathcal{D}'} \Big\| \big(\mathrm{TP},\ \mathrm{FP},\ \mathrm{FN},\ \mathrm{TN}\big) - \big(\mathrm{TP}',\ \mathrm{FP}',\ \mathrm{FN}',\ \mathrm{TN}'\big) \Big\|_1 .$$

Thus, as one trajectory can change at most 20 *3-gram, count parirs, we get*

$$\Delta_{quad} = 2 \cdot 20 = 40.$$

Proof. Let $\mathcal{D}$ be a dataset of trajectories and $\mathcal{D}'$ a neighbouring dataset differing in exactly one trajectory, which can change at most 20 3-grams.

Let U be the set of all possible 3-grams of some underlying transport etwork. Since S can only propose pairs in $\mathcal{U}$, we regard $S \subseteq U$.

Use TP, FP, FN and TN as

$$TP = |S \cap G(\mathcal{D})|,\ FP = |S \backslash G(\mathcal{D})|, FN = |G(\mathcal{D}) \backslash S|, TN = |U \backslash (S \cup G(\mathcal{D}))|$$
$$TP' = |S \cap G(\mathcal{D}')|,\ FP' = |S \backslash G(\mathcal{D}')|, FN' = |G(\mathcal{D}') \backslash S|, TN' = |U \backslash (S \cup G(\mathcal{D}'))|.$$

Let F, be the difference of $G(\mathcal{D})$ and $G(\mathcal{D}')$, when $G(\mathcal{D}')$ adds a new trajectory not seen by $G(\mathcal{D})$ and remark $|F| \leqslant 20$. For each $g \in F$, exaclty one of the following cases holds:

1. $g \in S$
   - If $g \notin G(\mathcal{D})$ but $g \in G(\mathcal{D}')$
     $$TP' = TP + 1,\ FP' = FP - 1,\ FN' = FN, TN' = TN$$
   - If $g \in G(\mathcal{D})$ but $g \notin G(\mathcal{D}')$
     $$TP' = TP - 1,\ FP' = FP + 1,\ FN' = FN, TN' = TN$$
2. $g \notin S$
   - If $g \notin G(\mathcal{D})$ but $g \in G(\mathcal{D}')$
     $$TP' = TP,\ FP' = FP,\ FN' = FN + 1, TN' = TN - 1$$
   - If $g \in G(\mathcal{D})$ but $g \notin G(\mathcal{D}')$
     $$TP' = TP,\ FP' = FP,\ FN' = FN - 1,\ TN' = TN + 1$$

For each case we consider the quadruple $\Delta m = m' - m$, with $\left\| m \right\|_1 = |\Delta TP| + |\Delta FP| + |\Delta FN| + |\Delta TN|$ therefore for each case we get:

1. $g \in S$, $c \geqslant \tau$
   - If $g \notin G$ but $g \in G(\mathcal{D}')$:
     $$\Delta\mathbf{m} = (+1,\ -1,\ 0,\ 0), \quad \|\Delta\mathbf{m}\|_1 = |1| + |-1| = 2.$$
   - If $g \in G$ but $g \notin G(\mathcal{D}')$:
     $$\Delta\mathbf{m} = (-1,\ +1,\ 0,\ 0), \quad \|\Delta\mathbf{m}\|_1 = |-1| + |1| = 2.$$

2. $g \notin S$
   - If $g \notin G$ but $g \in G(\mathcal{D}')$:

$$\Delta\mathbf{m} = (0,\ 0,\ +1,\ -1), \quad \|\Delta\mathbf{m}\|_1 = |1| + |-1| = 2.$$

   - If $g \in G$ but $g \notin G(\mathcal{D}')$:

$$\Delta\mathbf{m} = (0,\ 0,\ -1,\ +1), \quad \|\Delta\mathbf{m}\|_1 = |-1| + |1| = 2.$$

Since $|F| \leqslant 20$ and every flip $g \in F$ changes m by at most $\left\|m\right\|_1$, summing over all g gives:

$$\begin{aligned}\Delta_{quad} &= \max_{D \sim D'} \left\|m' - m\right\|_1 \leqslant 2|F| \leqslant 40 \\ &= 40\end{aligned}$$

Thus, the sensitivity of the quadruple (TP, FP, FN, TN) is $\Delta_{quad} = 40$.

□

Using Lemma 6.2 it remains to show that Algorithm 10 is ε-DP.

Lemma 6.3 (MobiTrie is ε-DP).
Let D be any finite dataset of trajectories $\mathcal{T}$, each trajectory contributing at most 20 3-grams, and let Algorithm 10 be instantiated with privacy parameters $\varepsilon_1, \varepsilon_0$ and shares $r_1, r_2 > 0$ with $r_1 + r_2 = 1$. Then the overall mechanism is ε-DP. Let

$$\varepsilon_{\text{cnt}} = r_1\,\varepsilon_1, \quad \varepsilon_{f1} = r_2\,\varepsilon_1,$$

Then, due to the rejection sampling we get:

$$\varepsilon = 2\varepsilon_1 + \varepsilon_0,$$

making the MobiTrie algorithm ε-DP.

Proof. We invoke the following standard results:

Laplace Mechanism If a function f has global sensitivity Δf, then $\mathcal{M}(D) = f(D) + \text{Laplace}(\Delta f/\varepsilon)$ is ε-DP.

Sequential Composition The cascade of an ε_1-DP mechanism followed by an ε_2-DP mechanism (on the same data) is $(\varepsilon_1 + \varepsilon_2)$-DP.

Post-Processing Any further computation on the output of an ε-DP mechanism remains ε-DP.

Step 1: Noisy 3-gram counting. For each 3-gram g, let

$$c_g(D) = \left|\{\mathcal{T} \in D : g \text{ is in } \mathcal{T}\}\right|.$$

Since one trajectory $\mathcal{T}$ changes each c_g by at most 20, $\Delta c_g = 20$. By Laplace,

$$\tilde{c}_g = c_g(D) + \text{Laplace}(20/\varepsilon_{\text{cnt}})$$

is ε_{cnt}-DP.

Step 2: Noisy Quadruple. Lemma 6.2 shows the quadruple (TP, FP, FN, TN) has sensitivity $\Delta_{cf} = 20$, for a prediction set S and ground truth set G. Adding independent noisy to the quadruple:

$$\begin{aligned}\tilde{TP} &= TP + \text{Laplace}(40/\varepsilon_{f1})\\ \tilde{FP} &= FP + \text{Laplace}(40/\varepsilon_{f1})\\ \tilde{FN} &= FN + \text{Laplace}(40/\varepsilon_{f1})\\ \tilde{TN} &= TN + \text{Laplace}(40/\varepsilon_{f1}),\end{aligned}$$

gives us ε_{f1}-DP.

By post-processing, any further computation of an ε-DP mechanism is still ε-DP. Therefore,

$$\tilde{F}1 = \frac{2\tilde{TP}}{2\tilde{TP} + \tilde{FP} + \tilde{FN}}$$

still remians ε_{f1}-DP.

Step 3: Rejection sampling. As the rejection sampling process is dependent on the F_1 score, and the F_1 score is ε_{f1}-DP, by sequential composition, any test on $\{\tilde{c}_g\}$ and $\widetilde{F}_1$ is $(\varepsilon_{\text{cnt}} + \varepsilon_{f1})$-DP. The rejection sampler of [17], when parameterized with ε_0 and $\varepsilon_1 = \varepsilon_{\text{cnt}} + \varepsilon_{f1}$, provides $\varepsilon = 2 \cdot \varepsilon_1 + \varepsilon_0$ differential privacy.

By sequential composition, by performing:

$$\varepsilon_1 = 0.5 \cdot (\varepsilon - \varepsilon_0)$$

and introducing shares r_1, r_2 with $r_1 + r_2 = 1$ and choosing

$$\begin{aligned}\varepsilon_{\text{cnt}} &= r_1 \cdot \varepsilon_1\\ \varepsilon_{f1} &= r_2 \cdot \varepsilon_1\end{aligned}$$

all differentially private steps add up to

$$\begin{aligned}\varepsilon &= 2 \cdot (r_1 \cdot \varepsilon_1 + r_2 \cdot \varepsilon_1) + \varepsilon_0\\ &= 2 \cdot (\varepsilon_{\text{cnt}} + \varepsilon_{f1}) + \varepsilon_0\\ &= 2 \cdot \varepsilon_1 + \varepsilon_0\\ &= 2 \cdot (0.5 \cdot (\varepsilon - \varepsilon_0)) + \varepsilon_0\\ &= \varepsilon - \varepsilon_0 + \varepsilon_0\\ &= \varepsilon\end{aligned}$$

provding an ε-DP algorithm.

□

Algorithm 10 MobiTrie: Create a differentially private trie

```
Require:
    Universe of all possible 3-grams $\mathcal{U}$
    Trajectory dataset $\mathcal{D}$
    F1-score threshold $\tau$
    Probability $\gamma$
    Privacy budgets $\varepsilon, \varepsilon_0$
    Budget shares $(r_1, r_2)$
 1: $T \geqslant \max\{ \frac{1}{\gamma} \ln \frac{2}{\varepsilon_0}, 1 + \frac{1}{e\gamma} \}$        ▷ #iterations for RejSamp
 2: $\mathit{3Gram}$ ←3Gram($\mathcal{D}$)        ▷ Algorithm 6
 3: $\mathit{3Gram}$ ←Counting($\mathit{3Gram}$)        ▷ Algorithm 7
    ▷ Initialize the $\varepsilon$ values with shares $r_1, r_2$, to ensure $\varepsilon$-DP
 4: $\varepsilon_1 \leftarrow 0.5 \cdot (\varepsilon - \varepsilon_0)$
 5: $\varepsilon_{cnt} \leftarrow r_1 \cdot \varepsilon_1$
 6: $\varepsilon_{f1} \leftarrow r_2 \cdot \varepsilon_1$

    ▷ Create union of $\mathit{3Gram}$ and $\mathcal{U}$
 7: for all 3-gram in $\mathcal{U}$ do
 8:     if 3-gram $\notin$ $\mathit{3Gram}$ then
 9:         $\mathit{3Gram}$ ←$\mathit{3Gram}$ $\cup$ (3-gram, 1.0)
10:     end if
11: end for

    ▷ Rejection sampling until the trie meets the F1-score threshold
12: for $1, \ldots, T$ do        ▷ ...for at most $T$ rounds
13:     tmp-3Gram ←$\mathit{3Gram}$        ▷ Create copy of $\mathit{3Gram}$
14:     for all (n, cnt) in tmp-3Gram do
15:         cnt ←cnt + Laplace(0, $\frac{k}{\varepsilon_{cnt}}$)        ▷ Noise the counts
16:     end for
17:     $\mathit{DP\text{-}3Gram}$ ←Select(tmp-3Gram, $\varepsilon_{cnt}$)        ▷ Algorithm 8
18:     Trie.build($\mathit{DP\text{-}3Gram}$)        ▷ Build the trie using $\mathit{DP\text{-}3Gram}$
19:     (TP, FP, FN, TN) ←Quadruple($\mathit{3Gram}$, $\mathit{DP\text{-}3Gram}$, $\mathcal{U}$)        ▷ Algorithm 9
20:     (TP, FP, FN, TN) ←(TP, FP, FN, TN) + Laplace(0, $\frac{40}{\varepsilon_{f1}}$) ▷ Noise TP, FP, FN, TN
21:     F1 ←$\frac{2 \cdot TP}{2 \cdot TP + FP + FN}$
22:     if $F1 \geqslant \tau$ then
23:         return DP-Trie        ▷ A Trie was found
24:     end if
25:     flip a $\gamma$-biased coin s.t. with prob. $\gamma$: return $\perp$
26: end for
27: return $\perp$        ▷ No Trie was found
```

7

Experiments

This chapter evaluates *MobiTrie* (Algorithm 10) by analysing its utility through multiple performance metrics. In addition to traditional machine learning metrics, mainly the F1-score, precision, recall, accuracy, Jaccard index, and the false negative rate (FNR), we incorporate the fitness score, which is a metric used in process mining that quantifies how well the constructed model (e.g., a trie) can replay the behaviour of the underlying dataset. A fitness of 1.0 indicates perfect replayability of all trajectories, while lower values reflect that less behaviour can be replayed [20].

In addition to these performance metrics, we examine the relative mean error rate of subsequences constructed from the trie. This metric is proposed by [6, 7, 9, 25, 26] and measures the average number of errors that subsequences of varying lengths have compared to the underlying dataset.

For a rich evaluation, we simulate the behaviour of $10,000$ and $1,000,000$ simulated users, using Algorithm 5, showing that our proposed algorithm performs best the more data is available. Additionally, we evaluate the MSNBC dataset [15], used by Chen et al. [6], as this approach is similar to ours, providing better comparability.

7.1 Research Questions

To estimate if the experiments reflect that the implementation meets our requirements, the following research questions are considered:

RQ1 Is the utility of the ε differentially private trie sufficient for further analysis? Is the underlying data correctly replayable?

RQ2 Are behaviours introduced not present in the underlying dataset, or is the underlying infrastructure preserved?

RQ3 Are the reconstruction errors in generated trajectories within reasonable bounds?

7.2 Experimental Setup

All experiments were executed on a Nvidia A100 with 40 GB of GPU memory, an AMD EPYC 7513 32-core Processor, and 512 GB of system RAM. Two evaluation datasets were generated using Algorithm 5, simulating $10,000$ and $1,000,000$ users. Additionally, we evaluate the MSNBC dataset [15], which was used by Chen et al. [6], for better comparability with this work, as it is closely related to our approach.

Datasets	$\|\mathcal{D}\|$	$\|I\|$	$\max\|S\|$	$avg\|S\|$
Berlin: 10,000	10,000	2,951	112	43.37
Berlin: 1,000,000	1,000,000	2,994	135	43.7246
MSNBC	989,818	17	14,795	5.7

Table 7.1: Experimental dataset characteristics.

Table 7.1 summarizes the characteristics of the datasets, where $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$, $|I|$ is the number of unique locatins in $\mathcal{D}$, $\max|S|$ is the maximum length of trajectories in $\mathcal{D}$, $avg|S|$ is average length of all trajectories in $\mathcal{D}$. For each of the simulated datasets, we use the graph of all possible paths to get the universe of 3-grams. As the MSNBC dataset shows the visiting behaviour of users at MSNBC.com, no publicly available data exists for the universe. Thus, we consider all 17 unique locations and create all possible 3-gram combinations. This set of combined 3-grams is the universe of all possibilities.

We evaluated *MobiTrie* (Algorithm 10) over the set of $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$ performing 100 independent runs per ε, for a total of 500 runs per dataset. As a baseline, we consider a 3-gram prefix tree containing all 3-grams found in the underlying dataset $\mathcal{D}$. Therefore, no unseen 3-grams from the universe $\mathcal{U}$ are included as we assume that the baseline is the perfect trie showing the movement patterns solely given in $\mathcal{D}$. The following algorithmic parameters were held constantly:

- Budget shares $r_1 = 0.95,\ r_2 = 0.05$,
- Privacy budget $\varepsilon_0 = 0.01$,
- Probability $\gamma = 0.01$, and
- Acceptance threshold τ:
    1. $10,000$ simulated users: $\tau = 0.65$
    2. $1,000,000$ simulated users: $\tau = 0.7$
    3. MSNBC [15]: $\tau = 0.65$

The performance metrics were computed for all ε values and are defined as follows:

Precision $\frac{\text{TP}}{\text{TP+FP}}$, the proportion of correctly identified positives among all predicted positives.

Recall $\frac{\text{TP}}{\text{TP+FN}}$, the proportion of correctly identified positives among all actual positives.

F1-score $\frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision+recall}} = \frac{2\cdot TP}{2\cdot TP+FP+FN}$, the harmonic mean of precision and recall.

Accuracy $\frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$, the proportion of correct predictions over all instances.

Jaccard index $\frac{\text{TP}}{\text{TP+FP+FN}}$, measures similarity between predicted and actual positive sets.

False Negative Rate (FNR) $\frac{\text{FN}}{\text{FN+TP}}$, the proportion of actual positives missed by the model.

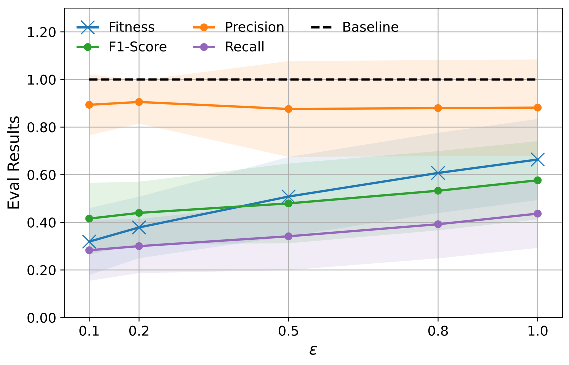
Fitness A process-mining metric quantifying replayability of observed sequences by the model, ranging from 0 (no replay) to 1 (perfect replay) [20]

7.3 Experimental Results - Performance Metrics

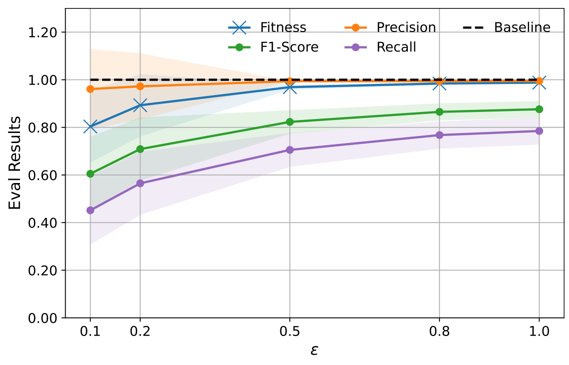
Figures 7.2, 7.3 and 7.4 show the results of the evaluated performance metrics. Black indicates the baseline, always 1.0, as it is a perfect trie of the underlying dataset. This also means that the FNR and FN of the baseline are always 0.0, as all 3-grams of the dataset are present in the trie. One of the first things we see is that with increasing ε, all metrics also increase, which is to be expected as less noise is introduced. The F1-score scales with recall and is less sensitive to precision, which is expected since the false positive (FP) rate is significantly lower than the true positive (TP) and false negative (FN) rates. Therefore, the precision has almost no impact on the F1 score. As FP is very low compared to the TP, the precision is close to 1.0, indicating that even though we use the universe of all 3-grams the significant selection (Algorithm 8) ensures that the tries generated stay close to the given dataset. Comparing Figure 7.2a and Figure 7.2b, we see that with $\varepsilon = 0.1$ the fitness is at approximate 0.3 for $10,000$ simulated users, whereas for $1,000,000$ users we achieve a fitness of 0.8. This indicates that with fewer data points, the algorithm experiences performance issues, resulting in tries with low utility. Even with $\varepsilon = 1.0$, the fitness only reaches 0.66 for $10,000$ simulated users, but 0.98 for $1,000,000$ users. This shows that the bound of using a maximum of 20 3-grams per trajectory is too much for fewer data points, as the amount of noise added strongly decreases the utility for this case. However, datasets with roughly $900,000$ trajectories preserve a good utility, as the fitness starts at approximately 0.9 for $\varepsilon = 0.1$, using the MSNBC dataset, fulfilling **RQ1**.

Figure 7.3 shows that the accuracy and the Jaccard index are close to the behaviour of the recall, which we expect as the false positives (FP) and true negatives (TN) should not be as high as the TP and FN. As FP and TN are close to zero, this shows that we do not introduce too much behaviour that is not possible in the dataset, thereby preserving the given infrastructure (**RQ2**). However, the fact that the accuracy and Jaccard index are lower than the FNR for $\varepsilon = \{0.1, 0.2\}$ over all three datasets indicates that we have more false negatives than true positives. This indicates that the amount of noise added to the counts results in a lower utility. With $\varepsilon = 0.5$, this changes, and the FNR falls under the accuracy and the Jaccard index. Nonetheless, this only happens for $1,000,000$ simulated users and the MSNBC dataset, showing that with fewer trajectories, the utility of *MobiTrie* is not improving as well, especially as the FNR remains greater than the accuracy and Jaccard index for $\varepsilon = 1.0$.

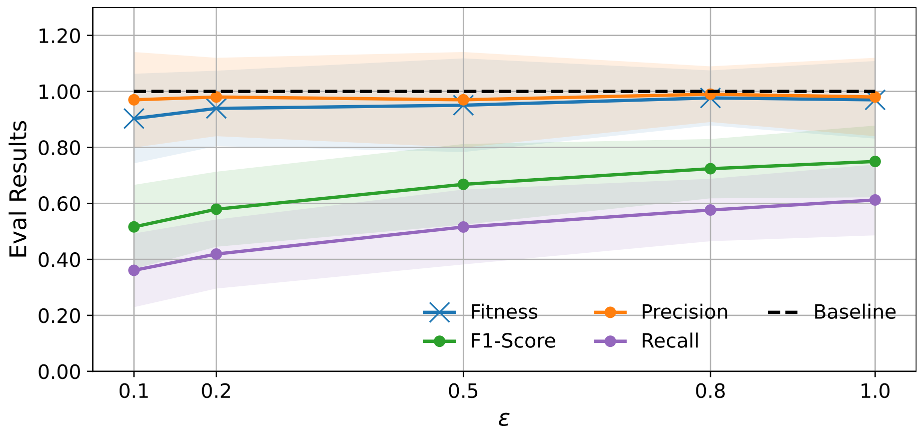
Having a look at Figure 7.4, we see that with increasing ε, the true positives (TP) increase and the false negatives (FN) decrease. This behaviour should happen as the amount of noise decreases with increasing ε, ensuring that more 3-grams from the dataset are used to build a trie. Additionally, we can see that the FP and TN are almost zero for datasets with approximately $900,000$. The data with the $10,000$ simulated users are not zero, but with increasing ε, both values decrease. Also, the behaviours observed in Figures 7.3 and 7.2 are mirrored by the TP and FN, which is to be expected as all metrics except the fitness depending on the TP and FN, as shown before.



(a) Evaluation results of the F1 score, fitness, precision, and recall, for 10, 000 simulated users (Algorithm 5).



(b) Evaluation results of the F1 score, fitness, precision, and recall, for 1, 000, 000 simulated users (Algorithm 5).

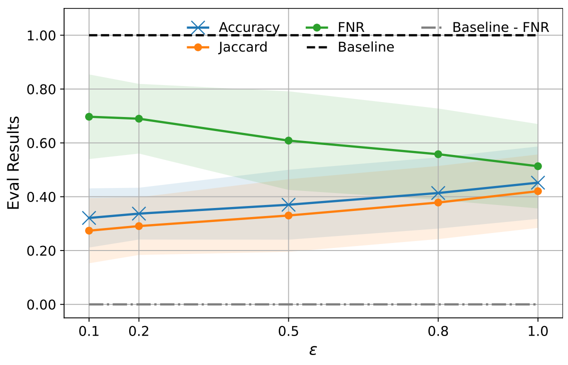


(c) Evaluation results of the F1 score, fitness, precision, and recall, for the MSNBC dataset [15].

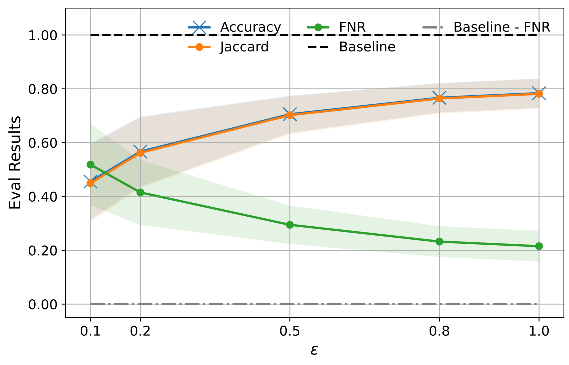
Figure 7.2: Evaluation results of the F1 score, fitness, precision, and recall. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value over all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is 1.0 for each of those metrics.

7.3.1 Detailed Evaluation

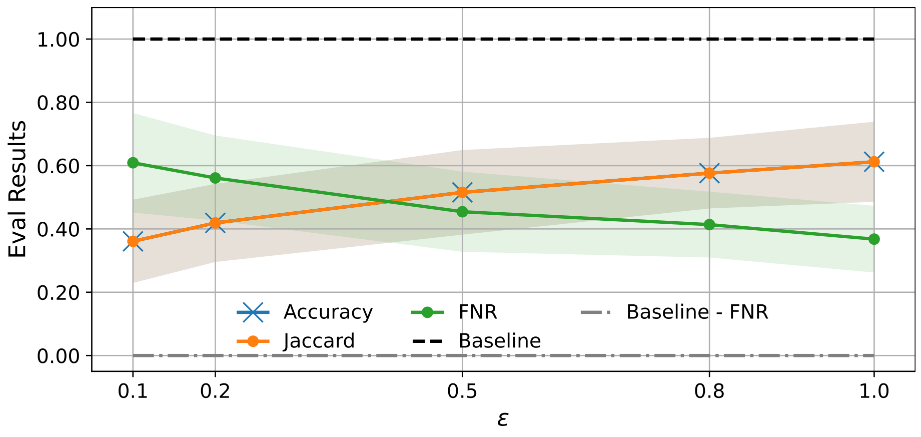
Tables 7.5, 7.6 and 7.7 show the detailed overview of the means of the evaluated performance metrics over all five ε values, for the three datasets. As can be seen in Tables 7.6 and 7.7, even for $\varepsilon = 0.1$ a fitness of 0.8 is achieved when simulating 900, 000 users. In contrast, the MSNBC dataset achieves a fitness of 0.90, showing that with a higher number of data points, the generated tries to achieve a good utility. Another important observation is that beyond $\varepsilon = 0.5$, the fitness reaches a value of > 0.95 for at least 900, 000 trajectories, showing that for those, almost perfect replayability is achieved while having reasonable good privacy. This significant increase in fitness is also evident when examining the true positives (TP) and false negatives (FN). While with $\varepsilon = 0.1$ the FN is at 5698.29 (Table 7.6), with $\varepsilon = 0.5$ the FN is almost halved, and the TP reaches 7698.84 coming near a perfect score of 11, 044. For the MSNBC dataset (Table 7.7), a similar behaviour is observed. The FN starts at



(a) Evaluation results of the Accuracy, Jaccard index, and the FNR, for 10,000 simulated users.



(b) Evaluation results of the Accuracy, Jaccard index, and the FNR, for 1,000,000 simulated users.

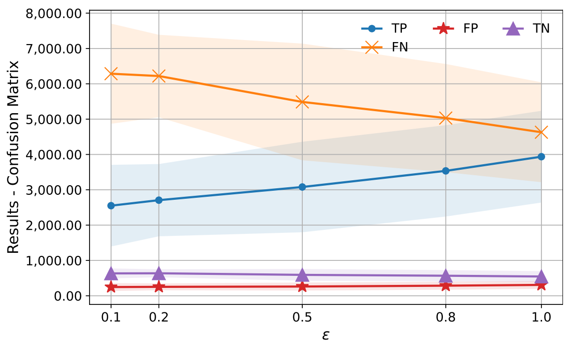


(c) Evaluation results of the Accuracy, Jaccard index, and the FNR, for the MSNBC dataset [15].

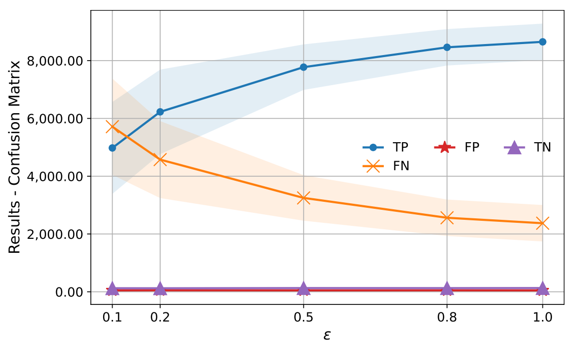
Figure 7.3: Evaluation results of the Accuracy, Jaccard index, and the FNR. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value over all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is 1.0 for the Accuracy and the Jaccard index. The grey line shows the baseline for the FNR, which is 0.0.

2641.59, for $\varepsilon = 0.1$, and falls to 1907.44. While the FN decreases, it is not as much of a decrease as observed in Table 7.6. This difference shows that MSNBC is not as suited for *MobiTrie* as our simulation is, which is expected as 365,435 of the 989,818 trajectories in the dataset have a length of 1, forcing our algorithm to discard 1/3 of the data. However, while the increasing and decreasing behaviour of the TP and FN is slower, compared to Table 7.6, the MSNBC dataset does not have any false positives or true negatives. This shows that even though we chose the set of all possible 3-grams of all unique identifiers as the universe, the dataset already uses all of those possibilities, ending in *MobiTrie* not drawing from the universe as that is already covered by MSNBC naturally.

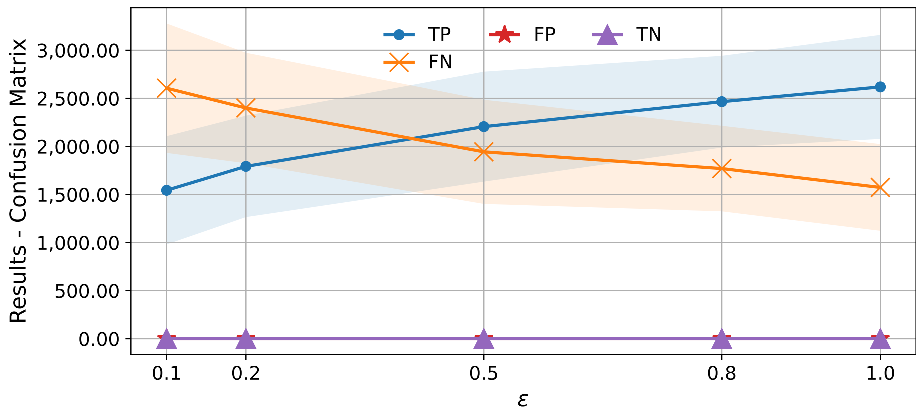
On the other hand, when looking at Table 7.5, it becomes clear that fewer data points result in tries with little utility, even for $\varepsilon = 1.0$, as the fitness only reaches 0.66. This



(a) Evaluation results of the TP, FP, FN, and TN for 10,000 simulated users.



(b) Evaluation results of the TP, FP, FN, and TN for 1,000,000 simulated users.



(c) Evaluation results of the TP, FP, FN, and TN, for the MSNBC dataset [15].

Figure 7.4: TP, FP, FN, and TN evaluation results. Done for $\varepsilon = \{0.1, 0.2, 0.5, 0.8, 1.0\}$, 100 times for each value for all three datasets. The results show the mean and standard deviation of those 100 runs. Black shows the baseline, which is 11,044 for the TP. The green line indicates the addition of TP and FN, showing that they reach the baseline value and include all elements.

behaviour becomes clearer when looking at the TP and FN of Table 7.5. Here, we can see that the difference of the TP between $\varepsilon = 0.1$ and $\varepsilon = 1.0$ is 500, and for the FN, it is only 300. This slight difference in the values shows that the constructed tries are not improving enough to achieve a good utility, even when using higher ε values. This strongly indicates that *MobiTrie* is not performing well on smaller datasets.

The precision indicates that the significant selections ensure that mainly 3-grams from the dataset are used in the generation process rather than 3-grams from the universe. Nonetheless, there are precision values As the fitness reaches 0.9 with $\varepsilon = 0.2$, for 1,000,000 and remains above 0.9 for all ε values when using the MSNBC dataset, we can conclude that the utility of the generated tries is good (**RQ1**), as long as the number of trajectories is sufficient. Additionally, the recall, accuracy, and Jaccard index are closely related to each other as the FP and TN values are small compared to the TP and FN. This ultimately demonstrates that the tries constructed by *MobiTrie* (Algorithm 10) mainly contain 3-grams that exist in the given dataset, thereby preserving the existing infrastructure and thus showing that **RQ2** is fulfilled.

ε	Fitness	F1-score	Precision	Recall	Accuracy	Jaccard	FNR	TP	FN	FP	TN
0.1	0.319	0.415	0.89	0.282	0.321	0.273	0.697	2550.56	6286.10	112.76	633.46
0.2	0.378	0.439	0.91	0.300	0.337	0.290	0.689	2706.34	6220.49	97.068	636.46
0.5	0.508	0.479	0.88	0.341	0.370	0.330	0.608	3078.91	5487.24	113.83	591.68
0.8	0.607	0.532	0.88	0.392	0.413	0.378	0.557	3535.47	5030.68	114.49	567.28
1.0	0.663	0.576	0.88	0.436	0.452	0.420	0.513	3937.15	4629.00	111.83	545.57

Table 7.5: Mean values of the performance metrics at different ε budgets, for $10,000$ simulated users. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us.

ε	Fitness	F1-score	Precision	Recall	Accuracy	Jaccard	FNR	TP	FN	FP	TN
0.1	0.803	0.605	0.96	0.451	0.455	0.449	0.5184	4978.04	5714.27	23.15	122.81
0.2	0.892	0.708	0.97	0.565	0.566	0.562	0.4150	6227.56	4574.98	21.51	121.72
0.5	0.968	0.822	0.99	0.705	0.705	0.702	0.2947	7774.16	3248.84	18.32	127.86
0.8	0.984	0.865	0.99	0.767	0.766	0.764	0.2324	8461.16	2561.84	20.03	126.48
1.0	0.987	0.876	0.99	0.784	0.783	0.781	0.2152	8650.39	2372.61	20.88	128.83

Table 7.6: Mean values of the performance metrics at different ε budgets, for $1,000,000$ simulated users. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us.

ε	Fitness	F1-score	Precision	Recall	Accuracy	Jaccard	FNR	TP	FN	FP	TN
0.1	0.903	0.516	0.97	0.360	0.360	0.360	0.609	1543.81	2605.85	0.0	0.0
0.2	0.939	0.579	0.98	0.419	0.419	0.419	0.561	1792.48	2399.96	0.0	0.0
0.5	0.950	0.668	0.97	0.515	0.515	0.515	0.454	2205.79	1943.87	0.0	0.0
0.8	0.976	0.723	0.99	0.576	0.576	0.576	0.413	2465.42	1769.80	0.0	0.0
1.0	0.969	0.749	0.98	0.612	0.612	0.612	0.367	2619.10	1573.34	0.0	0.0

Table 7.7: Mean values of the performance metrics at different ε budgets, for the MSNBC dataset [15]. For the FP, FN, and TN, we expect decreasing values with increasing ε. For all other metrics, an increasing value is expected by us.

7.4 Experimental Results - Mean Error Rate

The mean error rate is a metric proposed in [6, 7, 9, 25, 26] to evaluate how many errors are introduced when generating trajectories of different lengths. The utility of the counted 3-gram Q is measured by the relative error of its answer on the trie $Q(\tilde{D})$ concerning the original dataset $Q(D)$. The relative error is formalised as follows:

$$\text{error}(Q(\tilde{D})) = \frac{|Q(\tilde{D}) - Q(D)|}{\max(Q(D), s)}.$$

Where s is a sanity bound that mitigates the effect of 3-grams with low counts [25, 26]. Following [6, 7] we set s to 0.1% of $|D|$. $Q(D)$ is simply the number of times a 3-gram Q occures in the dataset, with D and $\tilde{D}$ being the base and noisy count. For example, looking at Table 3.1, $Q = (L_1, L_3, L_4)$ returns 4. Furthermore, we divide all queries into five subsequences with different lengths $(4, 8, 12, 16, 20)$. For each subsequence, we generate $10,000$ random queries of sizes that are uniformly random between 1 and each of the five lengths. Each query item is uniformly randomly selected from the dataset $\mathcal{D}$ [6, 7]. In this chapter, we discuss the results for $\varepsilon = \{0.1, 0.2, 0.5, 1.0\}$, as $\varepsilon = 0.8$ only marginally deviates from the other results, the corresponding results can be found in Appendix A.3.

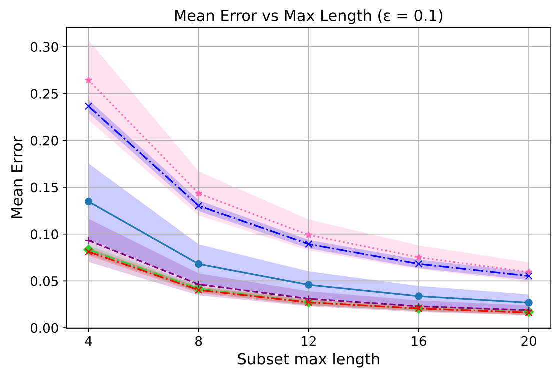
Overall, we can see in Figure 7.8 that with increasing ε, the error rates constantly decrease, showing that the proposed algorithm (Algorithm 10) reduces the noise level with increasing ε. However, looking at Figure 7.8a, we can observe that for $10,000$ simulated users, we achieve a reasonable error rate of 0.13 for $\varepsilon = 0.1$, at a subsequence length of 4. This shows that the low utility (fitness = 0.3, $F_1 = 0.4$) is not an influencing factor for the error rate. Nonetheless, the rate indicates that the noisy counts do not deviate significantly from the original counts, suggesting that the noise introduced primarily affects the utility of the tries. Furthermore, the error rates for $1,000,000$ simulated stays below 0.1 for all ε values over all subsequence lengths. On the other hand, the error rates of the MSNBC dataset exceed 0.2, even though it has the highest fitness of all three datasets. This again shows that the utility is not influencing the error rates. As the MSNBC datasets contain approximate $300,000$ trajectories of length 1, those are not covered by the 3-gram prefix trees constructed by *MobiTrie*. As the error rate randomly chooses a subsequence length between 1 and $(4, 8, 12, 16, 20)$, it happens that 1 is chosen. However, the corresponding trajectories (1/3 of the dataset) are not included in the prefix tree, increasing the error rate.

Nonetheless, for all three datasets, we can observe that the error rates are close to the error rates of the baselines, showing that the amount of noise added to the counts is not too much, ultimately showing that **RQ3** is fulfilled, no matter the dataset size.

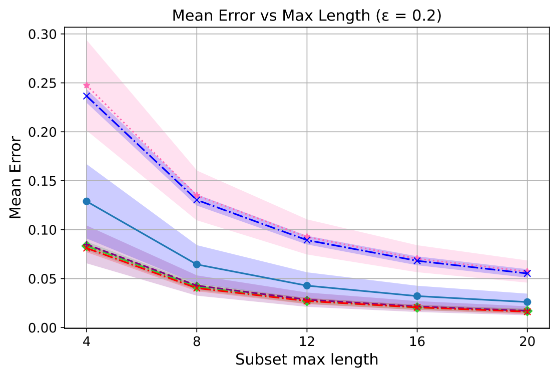
7.4.1 Detailed Evaluation

Tables 7.9, 7.10 and 7.11 report the mean and standard deviation of the relative error rates, as in Figure 7.8, for answering 3-gram queries Q on the ε-differentially private trie $\tilde{D}$, for varying subset lengths $(4, 8, 12, 16, 20)$ and privay budgets $\varepsilon = \{0.1, 0.2, 0.5, 1.0\}$. Looking at Table 7.11, we see that even though the ε values increase and Section 7.3 shows that the increase of ε has an impact on the utility, the error rate only marginally changes, as soon $\varepsilon \geqslant 0.2$. This behaviour is to be expected, as approximately one-third of the MSNBC dataset is discarded by our proposed algorithm. Therefore, the error rate cannot improve. For our assumptions regarding mobility data (Chapter 4) such discarding is expected, as those trajectories only contain one element, which is something not covered by us.

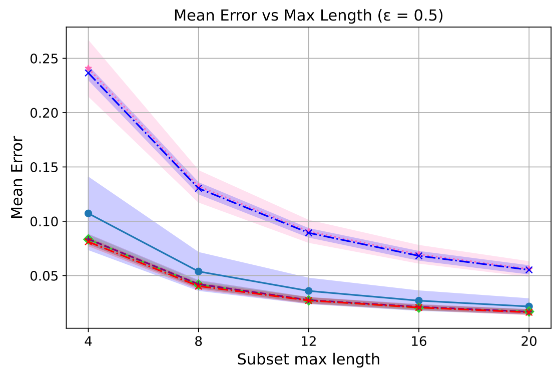
Nonetheless, the error rate is slightly better than the rates of Chen et al. [6], for $\varepsilon = 0.1$, as we achieve a rate of 0.05, whereas [6] achieves 0.07, for a length of 20. This



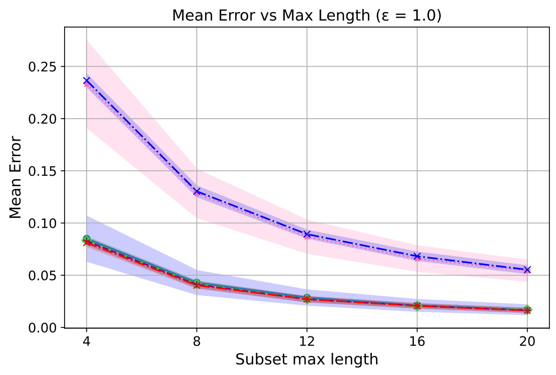
(a) Mean error of 100 runs using $\varepsilon = 0.1$.



(b) Mean error of 100 runs using $\varepsilon = 0.2$.



(c) Mean error of 100 runs using $\varepsilon = 0.5$.



(d) Mean error of 100 runs using $\varepsilon = 1.0$.

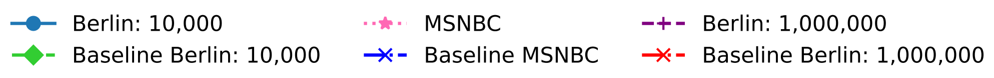


Figure 7.8: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = \{0.1, 0.2, 0.5, 1.0\}$, 100 times for each value. The results show the mean and standard deviation of those 100 runs. See Appendix A.3 for all plotted results. For visibility reasons, the results for $\varepsilon = 0.8$ can be found in Append A.3.

shows that even though the error rate is only marginally improving for higher ε values *MobiTrie* outperforms [6] for $\varepsilon = 0.1$. While we do not achieve improvements, [6] improves for $\varepsilon = 1.0$, achieving an error rate of 0.1 for a length of 4 and 0.025 for a length of 20. Therefore, higher ε values [6] achieve better error rates on the MSNBC dataset, while we perform better for lower ε values. Additionally, we can observe that by increasing the subsequence length from 4 to 8, the error rate halves, indicating that *MobiTrie* performs better on more extended sequences.

However, examining the error rates of the simulation reveals that the errors decrease with increasing ε, indicating that the amount of noise has a significant impact on the error rates. Looking at Table 7.9, we can observe that for $\varepsilon = 0.1$, the mean error rate is near 0.13, showing that even with low replayability (fitness = 0.3), the error rates are not influences. However, with a subsequence length of 8, the error rate drops to 0.6, halving the errors for a length of 4. This significant drop is observable for all ε values, showing that *MobiTrie* achieves better error rates for longer subsequences.

Looking at Table 7.10 we also observe this dropping behavior from Tables 7.9 and 7.11. Furthermore, the error rate remains below 0.1, showing that for $1,000,000$ simulated users and less discarded trajectories, as no trajectories with length $= 1$ exist, have a significant impact on the error rate.

Comparing Table 7.11 with Table 7.10, we observe that the simulation performs better than the MSNBC dataset. This shows that the discarding of trajectories has a significant impact on the error rates. Overall, with higher lengths, the error rates drop below 0.06, especially for lengths 20. We can observe that the error rate is half of the rate for the MSNBC dataset (Table 7.7). Additionally, the rates for $1,000,000$ are better than for $10,000$ simulated users, showing that the behaviour observed in Section 7.3 holds.

Furthermore, all error rates do not deviate from their corresponding baseline too much, ultimately showing that the errors are within reasonable bounds, fulfilling **RQ3**.

ε	`subset_length`	`mean_error`	`std_error`	ε	`subset_length`	`mean_error`	`std_error`
	4.0	0.1347	0.0398		4.0	0.1289	0.0733
	8.0	0.0680	0.0204		8.0	0.0644	0.0191
0.1	12.0	0.0458	0.0137	0.2	12.0	0.0428	0.0130
	16.0	0.0337	0.0102		16.0	0.0321	0.0099
	20.0	0.0267	0.0082		20.0	0.0260	0.0080
	4.0	0.1072	0.0332		4.0	0.0850	0.0215
	8.0	0.0537	0.0173		8.0	0.0430	0.0114
0.5	12.0	0.0358	0.0116	1.0	12.0	0.0286	0.0072
	16.0	0.0269	0.0089		16.0	0.0211	0.0056
	20.0	0.0215	0.0070		20.0	0.0169	0.0046

Table 7.9: Relative mean error rate by subsequence lengths for various ε budgets, for $10,000$ simulated users (Algorithm 5). For each ε budget, with increasing `subset_length` a decrease of the `mean_error` is expected.

7.5 Anserwering the Research Questions

As shown previously, the performance metrics indicate that the generated tries accurately replay the underlying dataset (fitness score of 0.91 for $epsilon = 0.1$, on MSNBC [15]) and are as precise as the baseline. Additionally, the relative error rate of all ε-differentially private tries compared to the non-private tries has a marginal deviation, showing that our implementation preserves the utility of the tries. As the metrics and the error rate show, tries generated by *MobiTrie* (Algorithm 10) preserve the utility, **RQ1** is fulfilled.

Figures 7.4a, 7.4b and 7.4c in combination with Tables 7.5, 7.6 and 7.7 show, that the tries have low false positives and true negatives. As those values are so low, it shows that all tries generated mainly use the 3-grams provided by the dataset $\mathcal{D}$ and less from the universe $\mathcal{U}$, fulfilling **RQ2**.

Since the relative error rate of ε-DP tries only marginally deviates from non-DP tries,

ε	subset_length	mean_error	std_error	ε	subset_length	mean_error	std_error
	4.0	0.0933	0.0223		4.0	0.0849	0.0187
	8.0	0.0463	0.00111		8.0	0.0429	0.0097
0.1	12.0	0.0309	0.0074	0.2	12.0	0.0284	0.0067
	16.0	0.0229	0.0056		16.0	0.0214	0.0050
	20.0	0.0187	0.0046		20.0	0.0170	0.0040
	4.0	0.0836	0.0042		4.0	0.0828	0.0033
	8.0	0.0420	0.0032		8.0	0.0411	0.0032
0.5	12.0	0.0274	0.0023	1.0	12.0	0.0269	0.0024
	16.0	0.0211	0.0023		16.0	0.0208	0.0020
	20.0	0.0166	0.0019		20.0	0.0163	0.0028

Table 7.10: Relative mean error rate by subsequence lengths for various ε budgets, for $1,000,000$ simulated users (Algorithm 5). For each ε budget, with increasing `subset_length` a decrease of the `mean_error` is expected.

ε	subset_length	mean_error	std_error	ε	subset_length	mean_error	std_error
	4.0	0.2642	0.0417		4.0	0.2474	0.0454
	8.0	0.1434	0.0228		8.0	0.1350	0.0248
0.1	12.0	0.0990	0.0159	0.2	12.0	0.0925	0.0173
	16.0	0.0752	0.0120		16.0	0.0703	0.0132
	20.0	0.0594	0.0097		20.0	0.0572	0.0107
	4.0	0.2407	0.0253		4.0	0.2331	0.0414
	8.0	0.1321	0.0143		8.0	0.1283	0.0231
0.5	12.0	0.0906	0.0100	1.0	12.0	0.0869	0.0158
	16.0	0.0690	0.0080		16.0	0.0659	0.0121
	20.0	0.0560	0.0067		20.0	0.0543	0.0100

Table 7.11: Relative mean error rate by subsequence lengths for various ε budgets, for the MSNBC dataset. With increasing `subset_length` the `mean_error` decreases.

we have a reasonable error rate over all ε for all subsequence lengths. Therefore, **RQ3** is fulfilled.

8 Conclusion

In this thesis, we design two algorithms: one to simulate the movement patterns of n users in a public transportation system and another to generate ε-differentially private 3-gram prefix trees of movement patterns. For the simulation, we use aspects of graph theory and distance measurements to generate data that mimics real-world behaviour. We apply techniques from mobility analysis and differential privacy to produce a trie that fulfills provable ε-DP and gives an overview of users' movement patterns.

As transit providers do not publish their mobility data (as discussed in Chapter 4), but mobility data is needed for a sound evaluation, we implemented a simulation based on publicly available GTFS data. For that, we model the data as a graph and use the well-known A* algorithm with the Haversine distance as the heuristic function to find the shortest path. This combination ensures a similar behaviour to the travel suggestions of mobile apps, as those also recommend the shortest routes. We introduce random hotspots (15 to 30 per simulation) to which users are more likely to travel so that the simulated movements appear as natural as possible. To keep this naturality, the hotspots are not guaranteed to be chosen as the destination, meaning that some simulated users take another random stop not being a hotspot. This is important, as not every user of a transport system travels to the point of interest, which leads us to take such behaviour into account.

For the *MobiTrie* algorithm (Algorithm 10 on page 41), we utilize Laplace noise in combination with rejection sampling ([11], Section 3.2) to generate ε-differentially private 3-gram prefix trees. Each trajectory in some given dataset, $\mathcal{D}$, can contribute a maximum of 20 3-grams, bounding the influence of all trajectories. As we bound the influence and use ε-DP mechanisms, we prove that *MobiTrie* achieves true ε-differential privacy (Lemma 6.3), enabling analyses of tries generated by us without violating the privacy of individuals.

We evaluate *MobiTrie* (Chapter 6) on a simulation of Berlins transport system (Algorithm 5) by simulating $10,000$ and $1,000,000$ users. As Chen et al. [6], who are closely related to *MobiTrie*, use MSNBC [15] to evaluate their approach, we also incorporate MSNBC into the evaluation. We evaluate known performance metrics from machine learning, mainly the precision, recall, F1-score, accuracy, Jaccard index, and the false negative rate (FNR). Additionally, we introduce the fitness from process mining as a metric to estimate the replayability of an underlying dataset [20]. All of those metrics show that the tries generated by *MobiTrie* have a good utility on datasets with approximate $900,000$ trajectories, as we achieve a fitness of 0.9 for $\varepsilon = 0.1$ on MSNBC. For the dataset with $10,000$ samples, the proposed algorithm does not achieve good fitness (< 0.6 for $\varepsilon = 1.0$) results, indicating that the introduced noise is too high for smaller datasets. However, all the $1,500$ generated prefix trees introduced marginally fewer false positives (FP) or true negatives (TN), indicating that the underlying infrastructure is not compromised too much.

Additionally to those performance metrics, we evaluate the relative mean error rate, proposed by [6, 7, 9, 25, 26]. This metric gives an overview of the errors introduced when creating subsequences of varying lengths based on a ε-DP trie. For the MSNBC dataset, we demonstrated that for $\varepsilon = 0.1$, *MobiTrie* algorithm outperforms Chen et al. [6], achieving approximately 20% better results at higher subsequence lengths. However, the structure of the MSNBC dataset results in less improvement for increasing ε values, different from Chen et al. [6]. Despite that, we observe a similar behaviour to Chen et al.: the longer the subsequences get, the fewer errors are introduced. Furthermore, the error rates of the two simulated datasets show that with increasing ε values, the errors decrease. This strongly indicates that the structure of the MSNBC dataset is a problem for *MobiTrie*. Additionally, we observe that for $10,000$ users, the error rate stays below 0.2 on $\varepsilon = 0.1$, showing that the utility of generated prefix trees does not influence the error rates. This observation indicates that the error rate alone is not enough to prove the utility of prefix trees on mobility data.

In summary, we demonstrate that *MobiTrie* fulfills its intention of generating a privacy-preserving 3-gram prefix tree. Furthermore, the algorithm preserves a good utility even for low ε values on datasets with $\geqslant 900.000$ trajectories. As we achieve good utility and error rates for $\varepsilon = 0.1$, tries generated by *MobiTrie* are suitable for further analysis, as the extracted information is a good representation of the underlying dataset, without violating the privacy of individuals.

8.1 Future Work

This section describes possible future work for each of the proposed algorithms.

Simulation Even though the simulation functions, it would give more realistic movements if the length and estimated time taken were both taken into account. For this, more information from the GTF files is required, and the heuristic function needs to be adjusted to use a combined weight of distance and time. Furthermore, differentiating between buses and metros could improve realism.

MobiTrie This work focuses on 3-grams for the trie generation. This is convenient for visualising transfers more effectively. However, Chen et al. [6] propose using varying length n-grams. Combining their approach with *MobiTrie*'s mechanisms such as including the whole universe of 3-grams $\mathcal{U}$, the significant selection (Algorithm 8) and rejection sampling (Algorithm 10), the error rate of the trie could be lowered, increasing the utility of generated tries and improving the error rate.

Bibliography

[1] Abul, O., Bonchi, F., and Nanni, M. Never Walk Alone: Uncertainty for Anonymity in Moving Objects Databases. In: *2008 IEEE 24th International Conference on Data Engineering*. 2008, pp. 376–385. DOI: 10.1109/ICDE.2008.4497446.

[2] Andrienko, G., Andrienko, N., Giannotti, F., Monreale, A., and Pedreschi, D. Movement data anonymity through generalization. In: *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*. SPRINGL '09. ACM, 2009, pp. 27–31. DOI: 10.1145/1667502.1667510.

[3] Beuth, P., Flüpke, Hoppenstedt, M., Kreil, M., Rosenbach, M., and Wilkin, R. Wir wissen, wo dein Auto steht. In: *Der Spiegel* (1), 2025. URL: https://www.ccc.de/de/updates/2024/wir-wissen-wo-dein-auto-steht.

[4] Cai, S., Lyu, X., Li, X., Ban, D., and Zeng, T. A Trajectory Released Scheme for the Internet of Vehicles Based on Differential Privacy. In: *IEEE Transactions on Intelligent Transportation Systems* 22(4):2408–2417, 2021. DOI: 10.1109/TITS.2020.2973451.

[5] Chen, J., Wang, W. H., and Shi, X. Differential Privacy Protection Against Membership Inference Attack on Machine Learning for Genomic Data. In: 2021. DOI: 10.1142/9789811232701_0003.

[6] Chen, R., Ács, G., and Castelluccia, C. Differentially Private Sequential Data Publication via Variable-Length N-Grams. In: *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS '12)*. Raleigh, NC, USA, Oct. 2012, pp. 638–649. DOI: 10.1145/2382196.2382263.

[7] Chen, R., Fung, B. C. M., and Desai, B. C. *Differentially Private Trajectory Data Publication*. 2011. arXiv: 1112.2020 [cs.DB]. URL: https://arxiv.org/abs/1112.2020.

[8] Chen, R., Fung, B. C., Mohammed, N., Desai, B. C., and Wang, K. Privacy-preserving trajectory data publishing by local suppression. In: *Information Sciences* 231:83–97, 2013. Data Mining for Information Security. DOI: https://doi.org/10.1016/j.ins.2011.07.035.

[9] Chen, R., Mohammed, N., Fung, B. C. M., Desai, B. C., and Xiong, L. Publishing set-valued data via differential privacy. In: *Proc. VLDB Endow.* 4(11):1087–1098, Aug. 2011. DOI: 10.14778/3402707.3402744. URL: https://doi.org/10.14778/3402707.3402744.

[10] Cunningham, T., Cormode, G., Ferhatosmanoglu, H., and Srivastava, D. Real-world trajectory sharing with local differential privacy. In: *Proceedings of the VLDB Endowment* 14(11):2283–2295, 2021. DOI: 10.14778/3476249.3476280.

[11] Cynthia Dwork, A. R. *The Algorithmic Foundations of Differential Privacy*. 2014. URL: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf.

[12] Dwork, C., McSherry, F., Nissim, K., and Smith, A. Calibrating Noise to Sensitivity in Private Data Analysis. In: *Theory of Cryptography*. Ed. by S. Halevi and T. Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284.

[13] Haydari, A., Chuah, C.-N., Zhang, M., Macfarlane, J., and Peisert, S. *Differential Privacy in Aggregated Mobility Networks: Balancing Privacy and Utility*. 2024. eprint: 2112.08487. URL: https://arxiv.org/abs/2112.08487.

[14] He, X., Cormode, G., Machanavajjhala, A., Procopiuc, C. M., and Srivastava, D. DPT: Differentially Private Trajectory Synthesis using Hierarchical Reference Systems. In: *Proceedings of the VLDB Endowment* 8(11):1154–1165, 2015. DOI: 10.14778/2850583.2850592.

[15] Heckerman, D. *MSNBC.com Anonymous Web Data*. UCI Machine Learning Repository. 1999. DOI: 10.24432/C5390X.

[16] Hu, H., Xu, J., On, S. T., Du, J., and Ng, J. K.-Y. Privacy-aware location data publishing. In: *ACM Trans. Database Syst.* 35(3), 2010. DOI: 10.1145/1806907.1806910. URL: https://doi.org/10.1145/1806907.1806910.

[17] Liu, J. and Talwar, K. *Private Selection from Private Candidates*. 2018. arXiv: 1811.07971 [cs.DS]. URL: https://arxiv.org/abs/1811.07971.

[18] Narayanan, A. and Shmatikov, V. Robust De-anonymization of Large Sparse Datasets. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. 2008, pp. 111–125. DOI: 10.1109/SP.2008.33.

[19] Rahimian, S., Orekondy, T., and Fritz, M. Differential Privacy Defenses and Sampling Attacks for Membership Inference. In: 2021. DOI: 10.1145/3474369.

[20] Rozinat, A. and Aalst, W. M. P. van der Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: *Business Process Management Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 163–176.

[21] Shaham, S., Ghinita, G., and Shahabi, C. *Differentially-Private Publication of Origin-Destination Matrices with Intermediate Stops*. 2022. eprint: 2202.12342. URL: https://arxiv.org/abs/2202.12342.

[22] Wang, H., Zhang, Z., Wang, T., He, S., Backes, M., Chen, J., and Zhang, Y. PrivTrace: Differentially Private Trajectory Synthesis by Adaptive Markov Models. In: *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023, pp. 1649–1666. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/wang-haiming.

[23] Wang, N. and Kankanhalli, M. *DPTraj-PM: Differentially Private Trajectory Synthesis Using Prefix Tree and Markov Process*. 2024. arXiv: arXiv:2404.14106.

[24] Wikipedia contributors *Great-circle distance — Wikipedia, The Free Encyclopedia*. [Online; accessed 02-June-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Great-circle_distance&oldid=1271272622.

[25] Xiao, X., Bender, G., Hay, M., and Gehrke, J. iReduct: differential privacy with reduced relative errors. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. SIGMOD '11. ACM, 2011, pp. 229–240. DOI: 10.1145/1989323.1989348. URL: https://doi.org/10.1145/1989323.1989348.

[26] Xiao, X., Wang, G., and Gehrke, J. *Differential Privacy via Wavelet Transforms*. 2009. eprint: 0909.5530 (cs.DB). URL: https://arxiv.org/abs/0909.5530.

[27] Yarovoy, R., Bonchi, F., Lakshmanan, L. V. S., and Wang, W. H. Anonymizing moving objects: how to hide a MOB in a crowd? In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology.* EDBT '09. ACM, 2009, pp. 72–83. DOI: 10.1145/1516360.1516370.

[28] Zhang, Y., Ye, Q., Chen, R., Hu, H., and Han, Q. Trajectory Data Collection with Local Differential Privacy. In: *Proceedings of the VLDB Endowment* 16(10):2591–2604, 2023. DOI: 10.14778/3603581.3603597.

A

Figures

In this chapter all figures used throughout the thesis are displayed. Those figures have a larger size, for a better overview and readability. Especially the figures showing an example simulation are easier to view, due to the size increase.

A.1 Figure of the Problem Statement

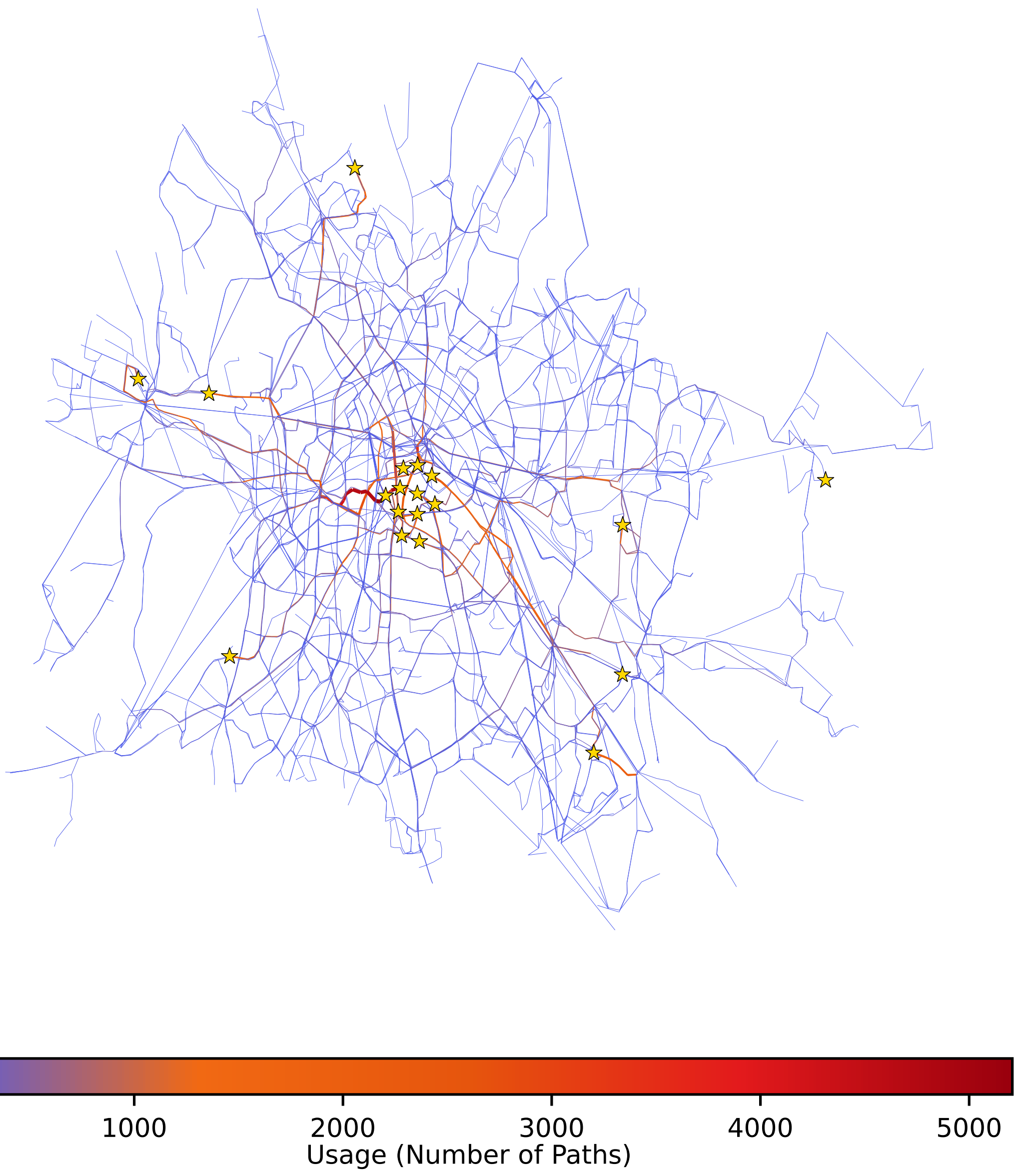


Figure A.1: A heat-mapped trajectory graph of Berlin's transit network. Star markers show endpoints. The path color gets redder the users take this path.

A.2 Figures of the Simulation

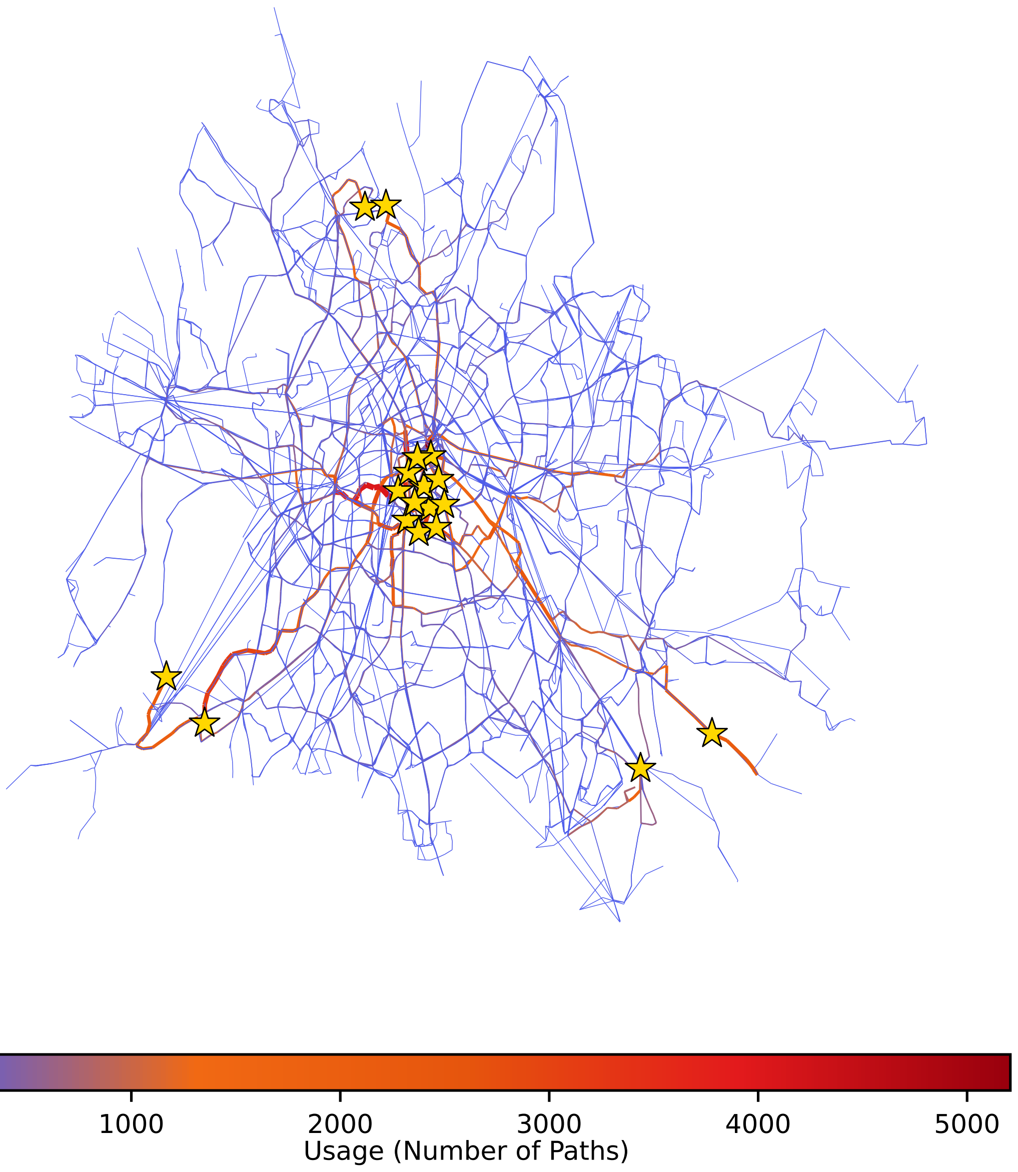


Figure A.2: Simulation using the hotspot generation from Algorithm 3 on page 24. Here in fullsize, for better readability.

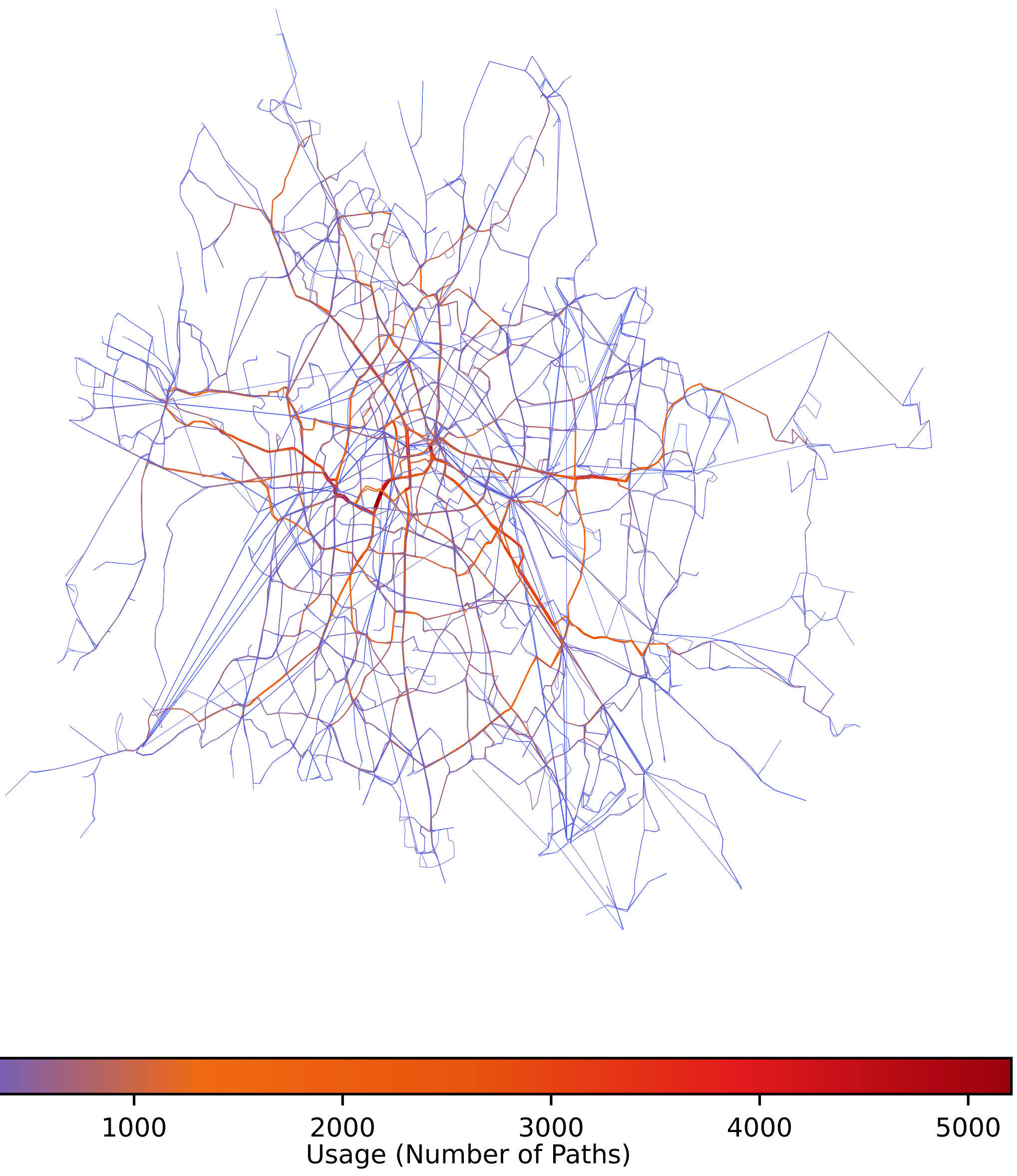


Figure A.3: Simulation without using the hotspot generation from Algorithm 3 on page 24. Here in fullsize, for better readability.

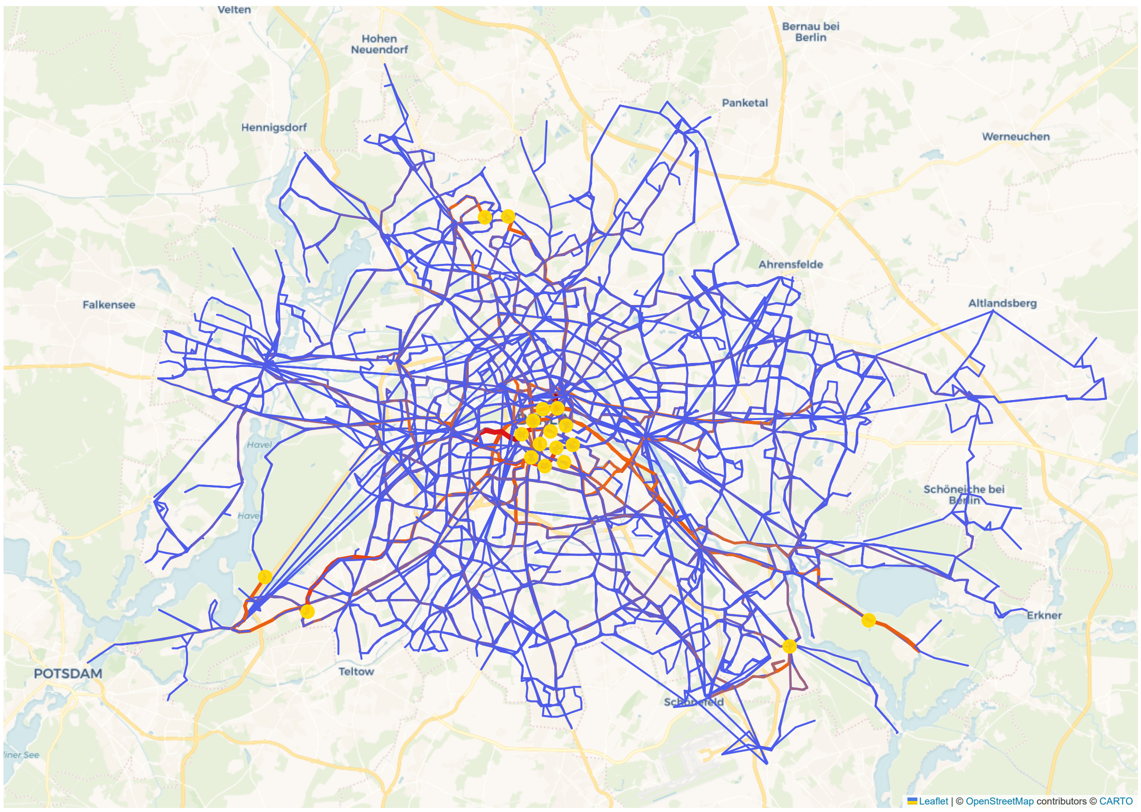


Figure A.4: Simulation of 100,000 users taking the public transport system in Berlin, displayed as a heatmap. The more a path is taken, the bigger and darker the connection is displayed. The trajectories are computed using the A* algorithm, with the Haversine distance as the heuristic function.

A.3 Evaluation Results - Mean Error Rate

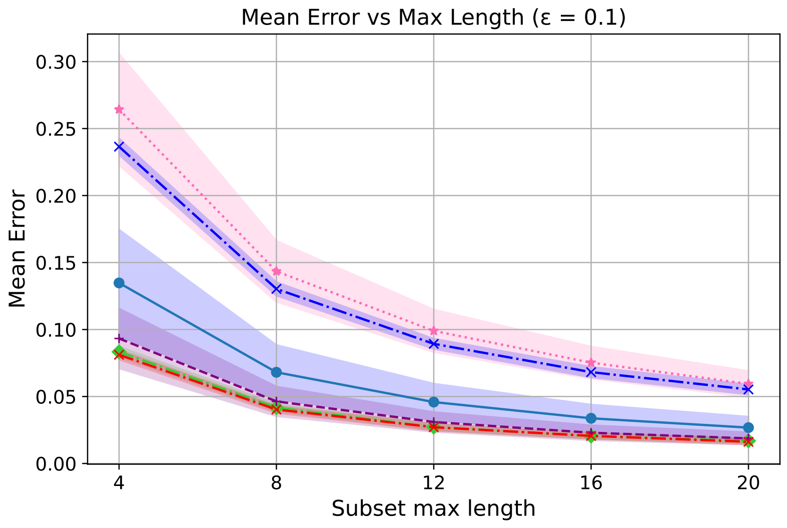


Figure A.5: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.1$ The result shows the mean and standard deviation of 100 runs.

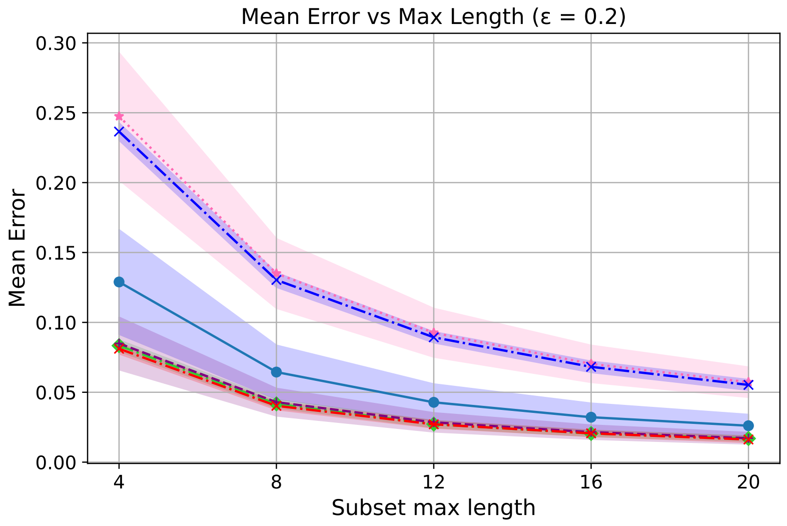


Figure A.6: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.2$ The result shows the mean and standard deviation of 100 runs.

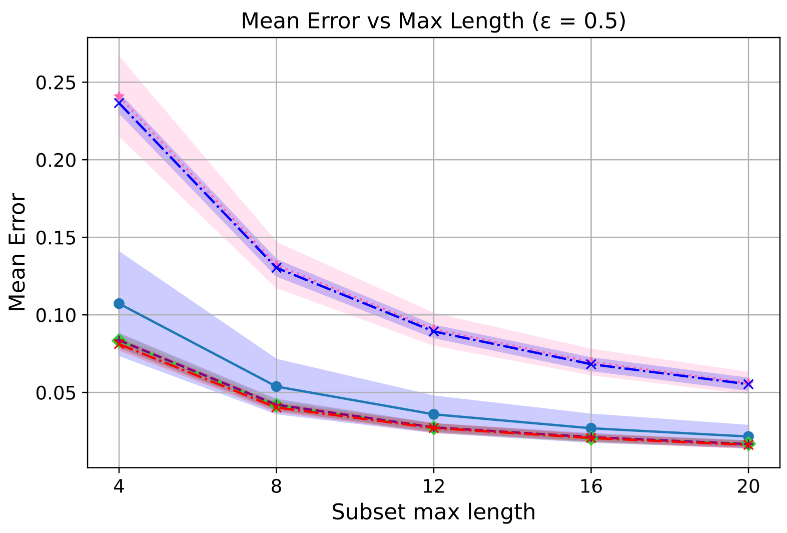


Figure A.7: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.5$ The result shows the mean and standard deviation of 100 runs.

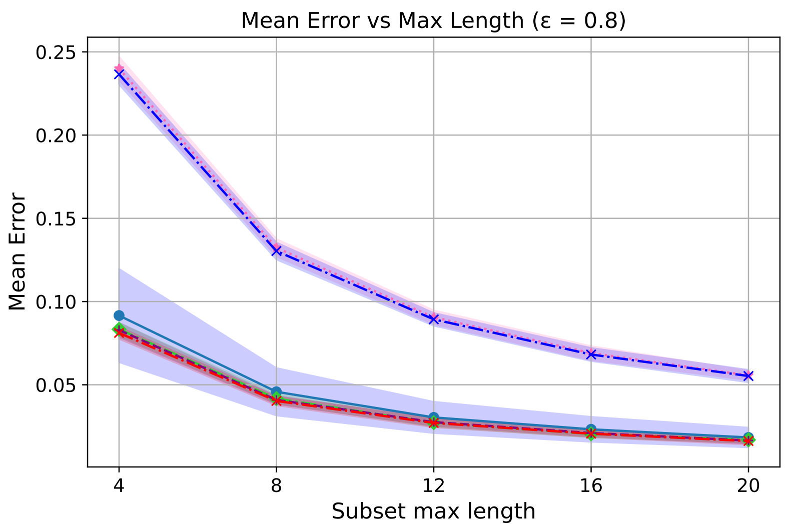


Figure A.8: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.8$ The result shows the mean and standard deviation of 100 runs.

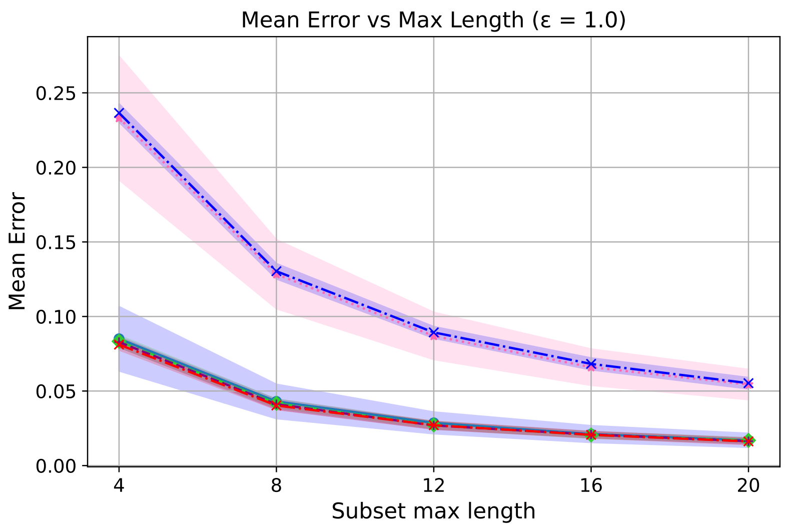


Figure A.9: Evaluation of the mean error, based on the length of trajectories. Here shown for $\varepsilon = 0.0$ The result shows the mean and standard deviation of 100 runs.

B

Possible Tries on MSNBC

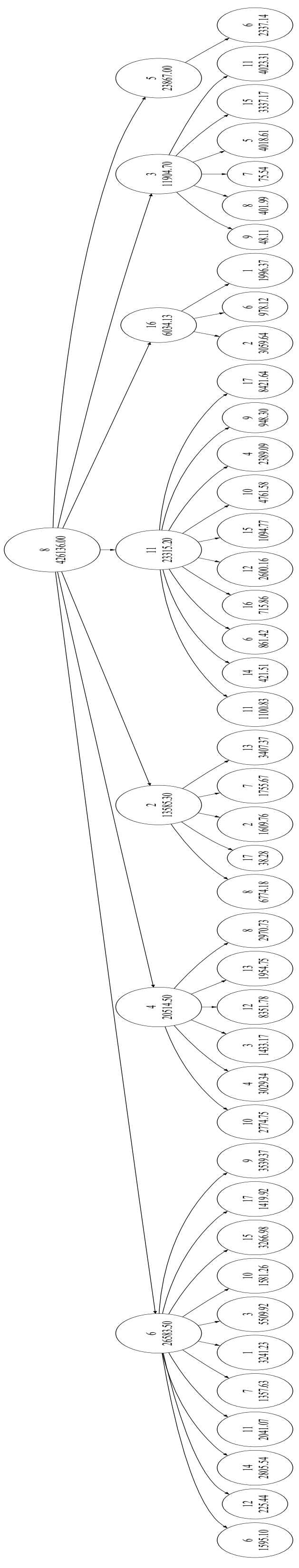


Figure B.1: Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.1$.

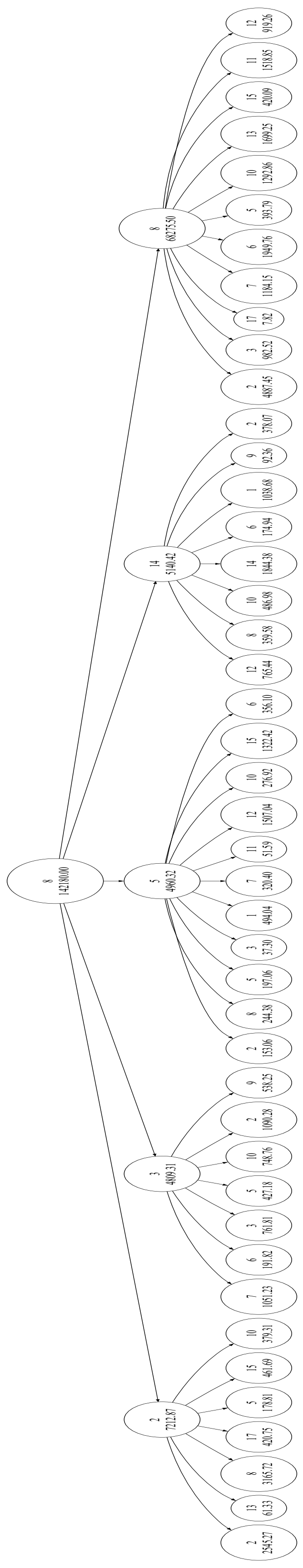


Figure B.2: Possible trie of the MSNBC dataset [15], using $\varepsilon = 0.5$.

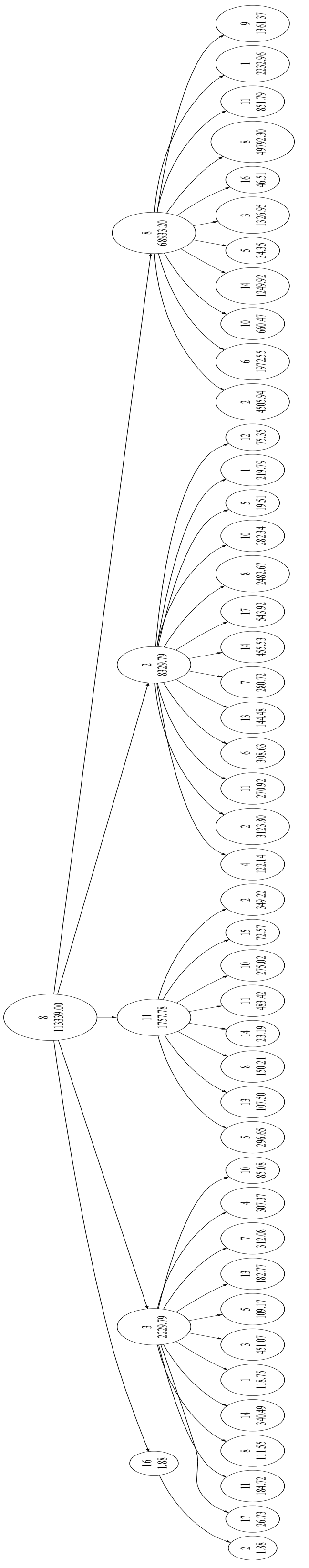


Figure B.3: Possible trie of the MSNBC dataset [15], using $\varepsilon = 1.0$.

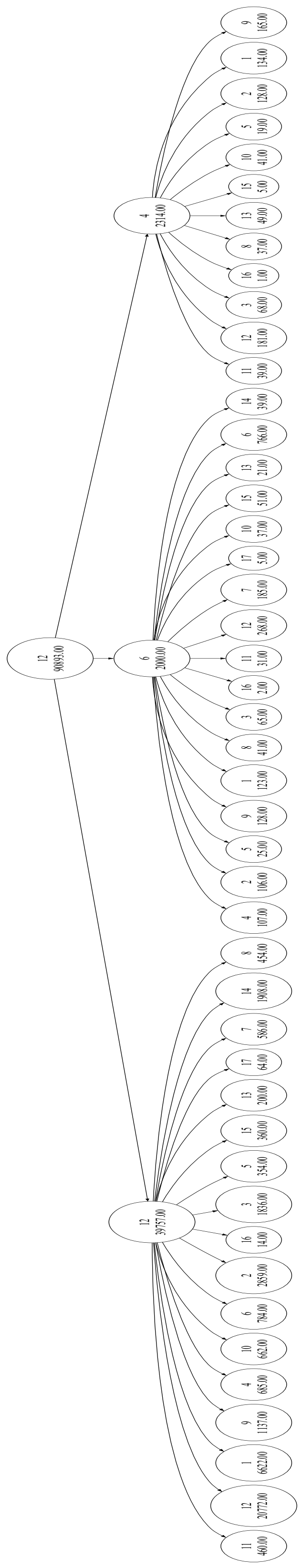


Figure B.4: Baseline trie of the MSNBC dataset [15]. Baseline means that no differential privacy mechanisms where used in the creation.