Utility-Optimized Synthesis of Differentially Private Location Traces

M. Emre Gursoy
Department of Computer Engineering
Koç University
Istanbul, Turkey
emregursoy@ku.edu.tr

Vivekanand Rajasekar
School of Computer Science
Georgia Institute of Technology
Atlanta, GA, USA
vivekraja07@gmail.com

Ling Liu
School of Computer Science
Georgia Institute of Technology
Atlanta, GA, USA
ling.liu@cc.gatech.edu

***Abstract*—Differentially private location trace synthesis (DPLTS) has recently emerged as a solution to protect mobile users' privacy while enabling the analysis and sharing of their location traces. A key challenge in DPLTS is to best preserve the utility in location trace datasets, which is non-trivial considering the high dimensionality, complexity and heterogeneity of datasets, as well as the diverse types and notions of utility. In this paper, we present OptaTrace: a utility-optimized and targeted approach to DPLTS. Given a real trace dataset D, the differential privacy parameter ε controlling the strength of privacy protection, and the utility/error metric Err of interest; OptaTrace uses Bayesian optimization to optimize DPLTS such that the output error (measured in terms of given metric Err) is minimized while ε-differential privacy is satisfied. In addition, OptaTrace introduces a utility module that contains several built-in error metrics for utility benchmarking and for choosing Err, as well as a front-end web interface for accessible and interactive DPLTS service. Experiments show that OptaTrace's optimized output can yield substantial utility improvement and error reduction compared to previous work.**

***Index Terms*—privacy, differential privacy, Internet of Things, privacy-preserving data analytics, trajectory data mining**

I. Introduction

As mobile devices and location-based services become increasingly ubiquitous, there is growing interest in analyzing and sharing information derived from mobile users' location traces. For example, Uber Movement shares anonymized data aggregated from billions of trips to help urban planning around the world [1]. Google's COVID-19 Community Mobility Reports share insights regarding movement trends over time by category (retail, grocery stores, pharmacies, transit stations, and so forth), which are also used in products such as Google Maps [2]. NYC Taxi and Limousine Commission shares taxi ride logs from New York City. Yet, the highly sensitive nature of mobile users' location traces gives rise to privacy risks when analyzing or sharing location data. Recent research has shown that many privacy attacks remain relevant despite aggregation or anonymization, such as stalking, trajectory reconstruction, de-anonymization, and membership inference attacks [3]–[9].

Differentially private location trace synthesis (DPLTS) has emerged as a solution to protecting mobile users' privacy while analyzing and sharing information derived from their traces [10]–[14]. In DPLTS, a generative synthesis system takes as input the dataset consisting of mobile users' real location traces (denoted D) and outputs a synthetic location trace dataset (denoted D_{syn}) which is syntactically and semantically similar to D, but consists of traces built while satisfying differential privacy. D_{syn} can then be used for in-house data analytics or for public release of statistics. DPLTS has two main privacy benefits. First, differential privacy provides a formal and robust privacy guarantee such that D_{syn} does not reveal the presence, absence or content of any real trace in D. Second, since the traces in D_{syn} are synthetic, they do not have one-to-one correspondence with any real individual; thus, re-identification and record linkage attacks are thwarted.

A central challenge in DPLTS, however, is how to best preserve the utility and statistical characteristics of D when synthesizing D_{syn}. This is a non-trivial challenge, considering the high dimensionality, complexity and heterogeneity of location trace datasets, e.g., varying dataset cardinality, trace length, trace duration, density, and sampling rate. In addition, there are endlessly many applications and statistics that could be derived from D_{syn}, such as travel time estimation, spatial density extraction and mobility pattern mining. Given that a different error metric or utility metric would be appropriate for each task, it is not feasible that a static DPLTS method preserves *all* utilities simultaneously.

Motivated by the above, this paper studies the following problem. Given a real trace dataset D, the differential privacy budget ε controlling the strength of privacy protection, and the utility/error metric Err of interest, we wish to optimize DPLTS such that output D_{syn} minimizes Err between D and D_{syn} while satisfying ε-differential privacy. Towards this goal, we design and develop the OptaTrace system which extends the AdaTrace system [11]. OptaTrace uses Bayesian optimization, a black-box optimization method, to find optimized parameters and budget distributions for AdaTrace's synopsis module which minimize error according to the given D, ε and Err. Furthermore, contributions of OptaTrace also include: (i) a utility module which contains several built-in error metrics to choose Err, as well as allowing the specification of a novel Err metric; and (ii) a front-end web interface for user-friendly and interactive DPLTS service. The user can upload their D, choose ε and Err through the web interface, as well as visually explore statistics regarding output D_{syn} or download D_{syn} to their local machine for further analysis.

OptaTrace provides a utility-targeted approach: If the utility metric Err is known ahead of time or can be approximated, OptaTrace's output D_{syn} can yield substantial utility improvement compared to untargeted (non-optimized) DPLTS approaches. We experimentally demonstrate the utility improvement of OptaTrace using three datasets, three ε values and four error metrics. Compared to the state-of-the-art AdaTrace system, OptaTrace outperforms AdaTrace in all experiments, and provides up to 50% reduction in utility loss. Our experiments also show that the optimized parameters are different for different D, ε and Err; which demonstrates the necessity of individual case-by-case optimization for targeted utility improvement.

The rest of this paper is organized as follows. In Section II, we review the location trace data model and differential privacy background. In Section III, we describe the OptaTrace system design. In Section IV, we give the implementation details of OptaTrace as well as a brief demonstration of its front-end web interface. Section V provides the results of our experimental evaluation. We summarize related work in Section VI and conclude in Section VII.

II. Data Model and Privacy Background

Consider a dataset $D = \{T_1, T_2, ..., T_{|D|}\}$ where each T_i corresponds to one mobile user's location trace. In order to protect the privacy of users' location traces, we enforce the popular notion of differential privacy [15], [16] as follows. Let $nbrs(D)$ denote the set of datasets neighboring D, such that for all $D' \in nbrs(D)$ the following holds: $(D - D') \cup (D' - D) = \{T\}$ where T denotes one location trace. Then, we say that a randomized algorithm $\mathcal{A}$ satisfies ε-differential privacy (ε-DP) if for all datasets D and $D' \in nbrs(D)$ and for all outcomes of the algorithm $S \in \textit{Range}(\mathcal{A})$:

$$\frac{\Pr[\mathcal{A}(D) = S]}{\Pr[\mathcal{A}(D') = S]} \leq e^{\varepsilon}$$

Here, ε is called the privacy budget, which determines the strength of privacy protection. Smaller ε gives stronger privacy.

Note that the above is a trace-level enforcement of differential privacy, i.e., it asserts that the outcome of the algorithm $\mathcal{A}$ will not enable an adversary to distinguish, beyond a probability controlled by ε, between two datasets D and D' that differ by a complete location trace T. This protects the complete location trace of a mobile user, and differs from DP perturbation of individual location points when the user is querying a location-based service [17], [18].

Differential privacy has three properties which are relevant and useful in the design of OptaTrace:

- *Sequential Composition*: For n algorithms $\mathcal{A}_1 \dots \mathcal{A}_n$ each satisfying DP with budget $\varepsilon_1 \dots \varepsilon_n$, the sequential execution of these algorithms on D satisfies $(\sum_{i=1}^{n} \varepsilon_i)$-DP.
- *Parallel Composition*: For two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfying ε_1-DP and ε_2-DP respectively, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are executed on disjoint subsets of D, the resulting execution satisfies $\max(\varepsilon_1, \varepsilon_2)$-DP.
- *Immunity to Post-Processing*: Let S denote the outcome of an ε-DP algorithm $\mathcal{A}$ executed on D, i.e., $\mathcal{A}(D) = S$. Then, any post-processing of S, including its use in a future algorithm or its public release, does not violate the ε-DP guarantee of S.

III. OptaTrace System

The goal of our OptaTrace system can be stated as follows: Given a real dataset D of actual location traces, the differential privacy budget ε, and the target utility/error metric Err, generate a synthetic location trace dataset D_{syn} such that ε-DP is satisfied and the utility loss between D and D_{syn} measured in terms of Err is minimized.

To achieve this goal, we designed the OptaTrace system as shown in Figure 1. It consists of four modules: synopsis module, optimization module, utility module and front-end web interface. In this section, we explain each module one by one.

OptaTrace extends the state of the art AdaTrace system [11] in three ways. First, OptaTrace includes a Bayesian optimization module for optimizing the parameter distribution according to given D, ε and Err. The optimization module iteratively searches for the optimized parameters that minimize Err, which are often different for different D, ε or Err. Second, OptaTrace includes a utility module which contains four categories of error metrics, so that the OptaTrace user can choose Err from existing metric categories or implement a new Err metric. The utility module of OptaTrace can also be used for benchmarking and evaluation of different D and D_{syn}. Third, OptaTrace provides a front-end web interface which enables OptaTrace users to seamlessly upload their D, choose their desired privacy level ε and metric Err through their favorite web browser. Preliminary statistics regarding the output D_{syn} can be obtained through OptaTrace's web interface, and D_{syn} can also be downloaded for further analysis.

A. Synopsis Module

The synopsis module of OptaTrace contains four features for extracting useful statistical information from D while satisfying differential privacy: density-aware grid $\mathbb{A}$, Markov model $\mathcal{M}$, trip distribution $\mathcal{R}$ and length distribution $\mathcal{L}$. These four features are then used by the trace generator (fifth component of the synopsis module) to generate synthetic traces which are added to D_{syn}. Below, we give brief descriptions of the four features and the trace generator. Full technical descriptions and privacy proofs can be found in [10], [11].

In order to satisfy ε-DP as a whole when extracting four features, OptaTrace makes use of DP's composition and post-processing properties. In particular, extracting the density-aware grid satisfies $(w_1 \times \varepsilon)$-DP, the Markov model satisfies $(w_2 \times \varepsilon)$-DP, the trip distribution satisfies $(w_3 \times \varepsilon)$-DP, and the length distribution satisfies $(w_4 \times \varepsilon)$-DP where the sum of the weights is: $\sum_{i=1}^{4} w_i = 1$. Thus, by sequential composition, the total of the four features satisfy ε-DP. The trace generator only uses the four features without modifying them or accessing the real dataset D, therefore ε-DP still holds due to immunity to post-processing.

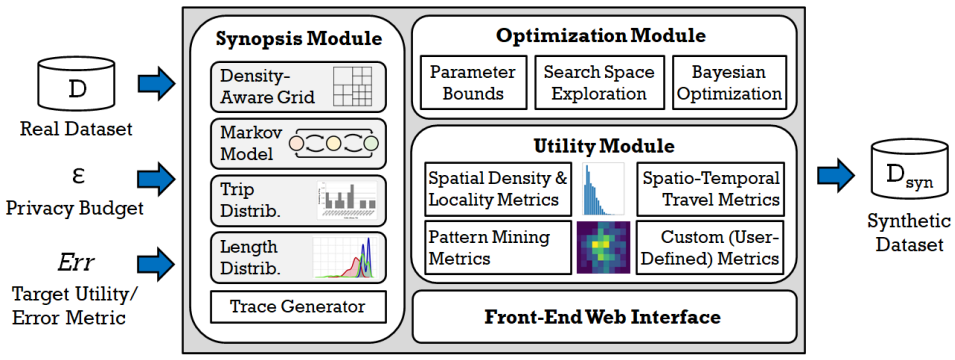


Fig. 1: OptaTrace system architecture

Density-Aware Grid $\mathbb{A}$: Accurately encoding the location space of D is the first step towards extracting useful statistics from D. We use a 2-dimensional grid structure to encode the location space of D, which is a common encoding strategy for location data. Yet, choosing an appropriate grid size and structure is non-trivial under DP and efficiency constraints. If the grid is too coarse (3x3), then each grid cell covers a large spatial area, and knowing that T visited a certain cell is uninformative. If the grid is too detailed (50x50), then there arise many empty cells with zero density, but noise must still be added to each of these cells to satisfy DP, which causes DP noise to overwhelm useful statistics, and inefficiency due to a large number of redundant empty cells.

In order to find a good balance, OptaTrace uses a density-aware grid structure $\mathbb{A}$ which adapts the number of cells that cover a geographic region according to the *density* of the region, i.e., the number of location readings in D that originate from that region. For low density regions, $\mathbb{A}$ places few large cells. For high density regions, $\mathbb{A}$ divides the region into many small cells. $\mathbb{A}$ is constructed in three steps: (1) Initially, an $N \times N$ uniform grid is laid in the geographic space covered by D, resulting in a total of N^2 cells. (2) For each cell, a density query is issued on D to retrieve how many normalized location readings exist in that cell. The answer to each density query is perturbed with randomized noise to satisfy DP. (3) Depending on their density, each of the original N^2 cells is either kept as is, or divided internally into smaller cells. Higher density implies more division, e.g., an extremely dense cell may be divided further into 6×6 smaller cells, whereas a medium density cell may be divided further into 2×2 smaller cells. The resulting grid by the end of step 3 is denoted $\mathbb{A}$.

An example density-aware grid $\mathbb{A}$ is given in Figure 2b. A 2×2 grid was initialized in step 1. The top-left cell was left without any further division due to low density. The top-right and bottom-left cells were each divided further into 2×2 cells due to having medium density. The bottom-right cell was divided into 3×3 cells due to having high density.

Markov Model $\mathcal{M}$: OptaTrace employs a Markov chain to model intra-trace mobility and movement behavior. Markov chains are a popular technique for mobility modeling, with many works showing that they are accurate in capturing urban mobility in real datasets and predicting users' next locations [19]–[21]. Our Markov model, denoted $\mathcal{M}$, contains:

- A set of Markov states: Each state corresponds to a cell C from the grid $\mathbb{A}$.
- Transition probabilities between states: For each pair of states C_i and C_j, there exists a transition probability for moving from state C_i to state C_j.

The transition probabilities are learned from the input real trace dataset D. Calibrated noise is added to each transition probability to satisfy DP. A sample Markov model is visualized in Figure 2c, where each state corresponds to a cell from grid $\mathbb{A}$, and the transition probabilities are written next to the transition arrows between each state.

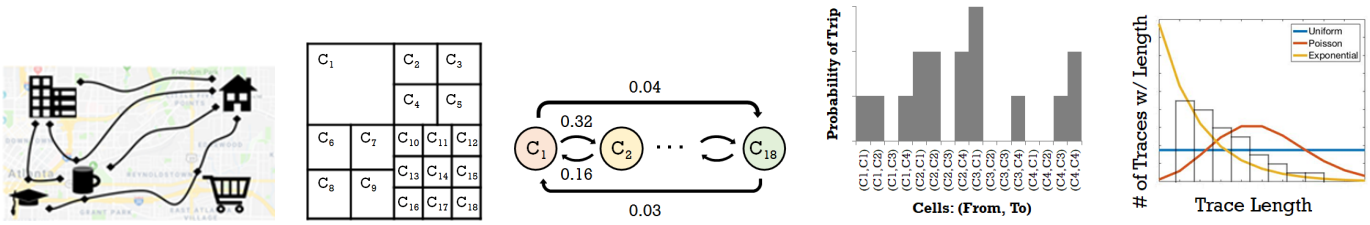
Trip Distribution $\mathcal{R}$: A mobile user's movement throughout the day often consists of several trips, e.g., home-work commute, lunch trip to a restaurant, trip to the gym after work, and so forth. Furthermore, real-life location trace datasets such as taxi or Uber traces often consist of a collection of trips. The trip distribution $\mathcal{R}$ in OptaTrace aims to preserve the joint association between the start-end locations of trips, which is useful for tasks including passenger demand analysis, taxi destination prediction, city planning, and so forth.

Let $T : C_i \rightsquigarrow C_j$ denote that location trace T starts its trip in cell C_i and finishes its trip in cell C_j. In essence, the trip distribution $\mathcal{R}$ is a probability mass function that contains one probability entry for each pair of cells $(C_i, C_j) \in \mathbb{A} \times \mathbb{A}$ that captures what percentage of traces in D make the trip $C_i \rightsquigarrow C_j$. Let $D_{C_i \rightsquigarrow C_j}$ denote the subset of D which consists of traces that make the trip $C_i \rightsquigarrow C_j$. Trip distribution $\mathcal{R}$ is:

$$\mathcal{R}_{D,\mathbb{A}}((C_i, C_j)) := \begin{cases} \frac{|D_{C_i \rightsquigarrow C_j}|}{|D|} & \text{for } (C_i, C_j) \in \mathbb{A} \times \mathbb{A} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Cardinalities $|D_{C_i \rightsquigarrow C_j}|$ and $|D|$ are perturbed with noise to satisfy DP. An example trip distribution is visualized in Figure 2d. (Note that this is a partial figure containing only 16 entries on the x-axis because of the space constraint. The actual trip distribution contains $\mathbb{A} \times \mathbb{A}$ entries.)

Length Distribution $\mathcal{L}$: There are likely to be multiple trips between a pair of cells $C_a \rightsquigarrow C_b$ and they may have varying length. OptaTrace learns the statistical length distribution for trips between $C_a \rightsquigarrow C_b$ as follows. First, the length of each trace $T \in D_{C_a \rightsquigarrow C_b}$ is measured. Second, a histogram is built



(a) (b) (c) (d) (e)

Fig. 2: Visualization of sample location traces and components of the synopsis module. (a) Visualization of real location traces. (b) Adaptive grid $\mathbb{A}$ with cells numbered C_1 to C_{18}. (c) Markov mobility model $\mathcal{M}$. (d) Trip distribution $\mathcal{R}$ (partially shown). (e) Calculation of length distribution $\mathcal{L}$ for a pair of cells $C_a \rightsquigarrow C_b$.

based on how many traces in $D_{C_a \rightsquigarrow C_b}$ have each length, e.g., 10 traces have length 2, 5 traces have length 3, and so forth. Third, three statistical distributions (Uniform, Poisson and Exponential) are initialized as potential candidates to represent the observed histogram. Finally, a goodness of fit test is used to determine which distribution best fits the observed histogram. The best fit distribution is stored in $\mathcal{L}$ and the rest are discarded. DP is enforced during the process of building the candidate distributions (one of which is eventually stored in $\mathcal{L}$) by adding noise to the parameters of the distributions.

We visualize the computation of the length distribution for one choice of $C_a \rightsquigarrow C_b$ in Figure 2e. The true length histogram is shown in bars. The three statistical distributions initialized while satisfying DP (Uniform, Poisson and Exponential) are shown with different colored lines. In this particular example, the goodness of fit test selects the Exponential distribution as the best fit, since its shape is closest to the shape of the histogram. Thus, the Exponential distribution would be stored in the length distribution for $C_a \rightsquigarrow C_b$.

Trace Generator: The trace generator is a synthesis algorithm which takes as input the previously computed four elements of the synopsis (density-aware grid $\mathbb{A}$, Markov model $\mathcal{M}$, trip distribution $\mathcal{R}$ and length distribution $\mathcal{L}$) and outputs a synthetic dataset of traces denoted D_{syn} with number of traces equal to cardinality of D. The trace generator does not modify the four existing synopsis elements or access the real dataset D. Since the synopsis elements already satisfy ε-DP as a whole, and since the execution of the trace generator performs only sampling and post-processing on the synopsis elements, the ε-DP guarantee still holds.

The trace generator generates each synthetic trace one by one, and adds them to D_{syn} upon generation. The steps to generate one synthetic trace denoted T_{syn} are as follows:

1) Draw a sample from $\mathcal{R}$ to determine the trip for T_{syn}. Let (C_{start}, C_{end}) denote the sampled trip.
2) Draw a sample from $\mathcal{L}$ to determine the length of T_{syn}. Let ℓ denote the sampled length.
3) Initialize T_{syn} with length ℓ, starting cell equal to C_{start}, and end cell equal to C_{end}.
4) To determine each of the intermediate locations in T_{syn}, perform a random walk on Markov chain $\mathcal{M}$. (Random walk is guaranteed to start in C_{start} and end in C_{end}.)

This process results in a synthetic trace consisting of exactly one trip, which is suitable when D or D_{syn} consists of Uber trips or taxi trips. Longer location traces (e.g., a mobile user's trace for one day or longer) are likely to contain multiple consecutive trips. In such situations, we extend the above process such that the C_{end} of the previous trip becomes the C_{start} of the next trip.

B. Utility Module

Recall that D denotes the input real dataset and D_{syn} denotes the output synthetic dataset. It is desired that D_{syn} preserves as much utility and statistical similarity to D as possible while satisfying the ε-DP guarantee. However, due to the noise addition in OptaTrace to satisfy ε-DP, D_{syn} will incur some utility loss compared to D. The goal of the utility module is to provide metrics for utility loss measurement.

Since location traces are inherently complex and utility in the location data analytics domain is a multi-faceted concept, there are many ways in which utility loss can be measured. Also, utility loss often depends on the end application and how D_{syn} will be used by the data analyst. For example, if D_{syn} will be used for building population density heatmaps, accurate representation of the location space and density preservation of D will be most important. In contrast, if D_{syn} will be used for analyzing taxi/Uber passenger demand, then preserving trip distributions will be most important. Consequently, the utility module of OptaTrace is designed to include a diverse set of built-in metrics for utility loss measurement, and also be extensible so that new metrics can be added in the future. Metrics in the utility module can be presented under 4 categories:

Spatial Density and Locality Metrics: Several geospatial analytics tasks rely on spatial densities and localities, such as Point-of-Interest analysis, spatial heatmaps, and location-based advertisement. Google's COVID-19 Community Mobility Reports [2] is a recent example requiring the preservation of spatial densities: each report highlights the percentage change in visits to places such as grocery stores, restaurants, parks and transit stations in a city when compared to a regular day before COVID-19. Utility metrics that measure error between D and D_{syn} in terms of spatial density and locality include: (i) error in computing the number of visits to a location using D_{syn} versus using D, (ii) error in determining location popularity rankings using D_{syn} versus D, e.g., error in restaurant

popularity rankings, (iii) error in computing answers to a range query workload using D_{syn} versus D, and so forth.

Spatio-Temporal Travel Metrics: Since location trace datasets often consist of taxi/Uber rides or daily commutes, analyzing aggregate trip features may yield not only a commercial advantage but also an urban planning advantage. For example, Uber Movement [1] provides a web interface for calculating average travel times between different neighborhoods, average road speeds at different times of day, etc. These statistics may be computed using D_{syn} rather than D to enforce formal privacy protection. Utility metrics that are suitable in measuring error in such an approach include: (i) error in computing average number of daily trips between two neighborhoods using D_{syn} versus D, (ii) error in estimating average street speed using D_{syn} versus D, (ii) error in estimating travel time between two neighborhoods using synthetic trip data in D_{syn} versus actual historical trip data in D, etc.

Pattern Mining Metrics: Pattern mining and pattern retrieval have been critical research problems in trajectory data mining. They have applications to not only human mobility patterns in urban environments but also to wildlife animals, e.g., finding seasonal migration patterns. Let $\mathcal{P}$ denote the results of pattern mining on D and $\mathcal{P}_{syn}$ denote the results of pattern mining on D_{syn}. The error between $\mathcal{P}$ and $\mathcal{P}_{syn}$ is measured by metrics including: (i) the set similarity between $\mathcal{P}$ and $\mathcal{P}_{syn}$, e.g., Jaccard similarity and F1 score, and (ii) the observation frequency of a pattern in $\mathcal{P}$ versus $\mathcal{P}_{syn}$.

Custom (User-Defined) Metrics: There can be metrics that are not covered by the categories and applications above. The design of OptaTrace allows the OptaTrace user to implement new, custom error metrics. The new metric can be a (weighted) combination of existing metrics, as well as a completely new metric inspired by a novel use case or unforeseen application of a location trace dataset.

C. Optimization Module

Recall from Section III-A that (w_1, w_2, w_3, w_4) are the four weight parameters of the OptaTrace system, and let Err denote the target error metric that is sought to be minimized. The goal of the optimization module can be stated as finding the values of w_1, w_2, w_3, w_4 such that:

$$\underset{w_1,w_2,w_3,w_4}{\operatorname{argmin}} \quad Err(D, D_{syn}) \tag{2}$$

That is, the optimization module aims to find the set of parameters for OptaTrace such that the output D_{syn} has lowest amount of error possible while satisfying ε-DP.

In order to achieve this goal, the optimization module uses *Bayesian optimization*, which is a class of machine learning-based methods for black-box function optimization [22], [23]. Its strategy is to treat the behavior of the synopsis module as a black-box function that needs to be optimized. First, it places a random prior regarding how the function behaves. Then, it gathers several evaluations of the function, e.g., executions with different parameter values (w_1, w_2, w_3, w_4) under the given D and ε. After observing the output of the function, i.e., the resulting error with the given set of parameters, it updates its belief regarding function behavior. Next, in each iteration the set of parameters is selected according to past observations and updated belief regarding which direction is best to explore for minimizing error. After several iterations, parameters converge to their optimized values which minimize $Err(D, D_{syn})$ under the given ε and D. Our use of Bayesian optimization can be explained in three consecutive steps.

(1) Specification of Parameter Bounds: In order to constrain the optimization search space, the initial step is to specify the bounds of each parameter that needs to be optimized. We specify the following constraints, which ensure that ε-DP is satisfied as a whole.

$$0 < w_1, w_2, w_3, w_4 < 1 \quad \text{and} \quad \sum_{i=1}^{4} w_i = 1 \tag{3}$$

(2) Search Space Exploration: Given the parameter constraints and the target error metric Err, we perform several *random explorations* to explore the search space, by executing the synopsis module with different random parameter sets and observing the resulting errors. This helps diversify the optimizer's prior beliefs and ensures that the search space is sufficiently probed before the actual optimization process begins. By default, we use 100 explorations.

(3) Iterative Bayesian Optimization: We execute several iterations of Bayesian optimization (by default, 100 iterations). In each iteration, the optimizer selects a set of parameters (w_1, w_2, w_3, w_4), executes the synopsis module, and observes the resulting error Err in terms of the given error metric. The observed error is used to update the optimizer's belief and informs the choice of parameters in the next iteration. As the number of observations grows, the optimizer becomes more certain which regions in the parameter space are more worth exploring. In time, the parameters converge to their optimized values which minimize Err.

D. Front-End Web Interface

We designed a front-end web interface for OptaTrace so that OptaTrace users can access and interact with OptaTrace through a user-friendly and interactive web interface. Our goals in designing the front-end web interface include:

- Accessible and easy-to-use privacy functionality for non-experts: Data analysts may be interested in adopting a privacy technology when performing location data analytics. However, off-the-shelf differential privacy (DP) tools are scarce, and they are often difficult to use for those who are not experts in DP. OptaTrace's web interface addresses this problem by providing an easy-to-use and interactive DP enforcement opportunity to data analysts.
- Differential privacy-as-a-service: Laws and regulations (such as GDPR) are increasingly restricting the storage of raw, sensitive user data. As a result, companies and businesses are turning towards innovative privacy protection methods so that user data can be privatized before being used or stored. OptaTrace's web interface can provide data privatization service in the following manner.

Consider that a company collects mobile users' location trace data, collectively denoted by D, as in Figure 1. Using the web interface, D is input to OptaTrace along with the privacy budget ε. The output dataset D_{syn} is downloaded and stored by the company; and the real data D is destroyed afterwards. As such, OptaTrace can serve as a differential privacy-as-a-service tool.

- Extensibility for client-server use and web hosting: The front-end design is suitable for client-server environments such that the OptaTrace software runs on a server machine, clients remotely connect to the server through the Internet, and they benefit from the OptaTrace privacy service. While we developed and tested the front-end web interface primarily in a single client environment, the client-server functionality may be extended in the future to enable OptaTrace be hosted on a central powerful web server, and clients use the OptaTrace service by connecting via their web browser.

IV. OptaTrace System Implementation

A. Implementation Details

The implementation of OptaTrace consists of three main parts: Web UI server, Python component, and Java component.

Web UI server provides the client-facing front-end web interface. It runs on Vue.js, a progressive open-source UI framework. It uses Vuetify, a Material Design component framework that provides a modern look, and Axios, an HTTP client for the browser. The Web UI server is used by the client to upload dataset D, choose privacy and optimization parameters, and download/analyze the output dataset D_{syn}. Axios is used to make REST API calls over HTTP to communicate the client's choices with the Python component. When the Web UI server needs to display information, it makes an HTTP request to the Python component, which in turn makes calls to the Java component.

Python component is written in Python and contains the optimization module of OptaTrace. It sits between the Web UI server and the Java component. It communicates with the Web UI server using REST API calls over HTTP that are handled using the Flask library. Upon receiving clients' commands and choices from the Web UI server, the Python component uses the Bayesian optimization library to iteratively perform parameter optimization. Each iteration requires back-and-forth communication with the Java component.

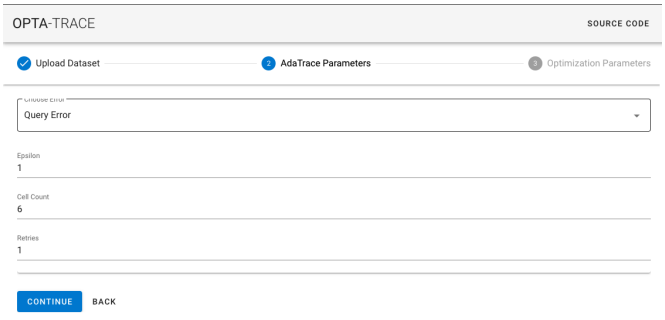
Java component contains the synopsis module and utility module of OptaTrace, written in Java language. It communicates with the Python component by using the Py4J library. Py4J enables Python programs running in a Python interpreter to dynamically access Java objects in a Java Virtual Machine [24]. Methods are called as if the Java objects resided in the Python interpreter and Java collections can be accessed through standard Python collection methods. Py4J also enables Java programs to call back Python objects. This allows the Python component to call methods from the Java codebase as if it were simply an extension of the Python component, which enables fast communication and data transfer between the Java component and Python component.

B. Execution and Brief Demonstration

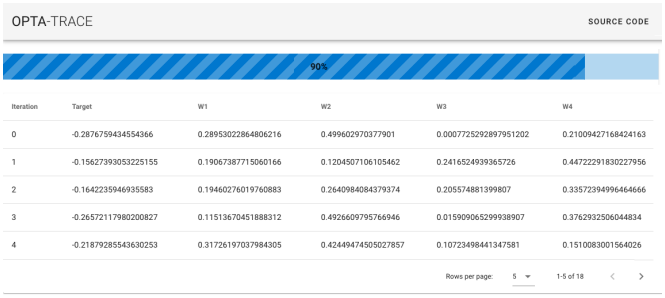
Combining the components listed in the previous section, the execution of OptaTrace has three phases. First, the Web UI server receives D, ε and related parameters from the OptaTrace user. Second, these are sent to the Python component which starts the optimization process. The Python component and the Java component communicate back-and-forth for many iterations of Bayesian optimization. In each iteration, the Python component instructs the Java component to run the synopsis module and utility module with a certain setting of parameters, observes the results, and updates the parameter settings for the next iteration. Third, after the optimization is complete, final results (D_{syn} and related statistics) are computed by the Java component, sent to the Python component, and then forwarded to the Web UI server. They are visualized and displayed to the OptaTrace user through the graphical web interface. In this section, we describe the three phases one-by-one and provide a screenshot for each phase in Figure 3.

Input Phase: In this first phase, the OptaTrace user is asked to provide the necessary inputs such as D, ε, Err metric. The phase consists of three substeps: Upload Dataset, AdaTrace Parameters, and Optimization Parameters. The Upload Dataset step asks the user to upload the real location trace dataset D. Once the user chooses an appropriate file for upload, the page shows a progressive loader that displays how much of the file has been uploaded. After the upload is complete, the user moves to the next step (AdaTrace Parameters). In the AdaTrace parameters step, the user is prompted to choose the Err metric to optimize, the privacy budget ε, the cell count for the first level of the adaptive grid, and the number of trials in each iteration of optimization to reduce the inherent randomness caused by DP noise. A screenshot from this step is provided in Figure 3a. Once these parameters are provided, the user moves to the last step (Optimization Parameters). In this step, the user is permitted to pick the number of random explorations and the number of guided explorations that the Bayesian optimization should take. Larger number of explorations cause optimization to take longer (i.e., longer wait time for the user) but will likely yield better-optimized results.

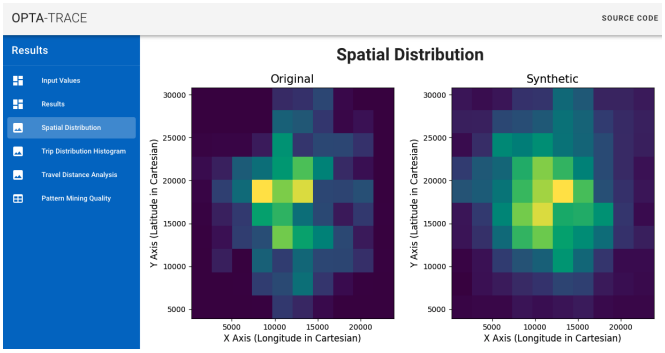
Computation and Optimization Phase: In this phase, the user views a page that displays the live-streamed results of the optimization process as it is running in the back-end system (Python component and Java component). A screenshot is provided in Figure 3b. Each step of the Bayesian optimization as well as the corresponding calculated error values are displayed to the user as soon as they are calculated. There is also a progressive loading bar that shows the user how much of the optimization process has been completed thus far. This is to inform the user about how many steps are completed, how many are left to finish, and accordingly, the user can estimate the completion time of optimization. Upon the completion of optimization, a button becomes available at the bottom of this page to take the user to the Results and Analysis phase.



(a) Input Phase



(b) Computation and Optimization Phase



(c) Results and Analysis Phase

Fig. 3: Screenshots from the OptaTrace front-end web interface showing the three main phases of user interaction.

Results and Analysis Phase: In this phase, the user is able to view and analyze the results of OptaTrace, e.g., D_{syn} and related statistics. A screenshot is provided in Figure 3c. It can be observed from the screenshot that this phase consists of six tabs on the left hand side. In the "Input Values" tab, the user can review the values that were chosen in the Input Phase which led to the current results. The "Results" tab displays the optimized set of parameters found using Bayesian optimization; furthermore, it lists the error values for D_{syn} computed using all of the built-in error metrics from the utility module (see Section III-B). This page also enables the user to download D_{syn}. The remaining four tabs are for analyzing and visualizing OptaTrace's output D_{syn} and comparing it with the original D. For example, the screenshot provided in Figure 3c is from the "Spatial Distribution" tab, where the user can see a spatial density heatmap of D and D_{syn} displayed side-by-side. The heatmaps are based on the x-y (or lat, long) coordinates of location traces, divided by 10x10=100 bins. In similar fashion, the "Trip Distribution" tab visualizes the trip distributions of D and D_{syn} side-by-side, the "Travel Distance Analysis" tab displays histograms of traces' travel distances in D and D_{syn} side-by-side, etc. The collective goal of these tabs is to provide an early visual insight on the impact of ε-DP on data utility. The tabs are extensible such that new visualizations may be added to the front-end source code.

V. Experimental Evaluation

A. Experiment Setup

Datasets: We experiment with three datasets that were also used in [10], [11]. Our first dataset is *Taxi*, which consists of GPS traces of taxis operating in the city of Porto, Portugal. The traces were made available as part of the Taxi Service Prediction Challenge at ECML-PKDD 2015 [25]. We extracted 15,000 taxi trips from the denser areas in the city to construct our Taxi dataset. Our second dataset is *Brinkhoff-20k*, which contains location traces of vehicles simulated using Brinkhoff's network generator for moving objects [26]. The map of Oldenburg, Germany was used to simulate movements of 20,000 vehicles and their locations were sampled at 15.6 second time intervals. Our third dataset is *Brinkhoff-4k*, which is a small sample consisting of 4,056 traces extracted from the Brinkhoff-20k dataset. The purpose of using both a large version and small version of Brinkhoff is to compare the behavior of errors and optimization on two semantically similar but cardinality-wise different datasets.

Competitors: We compare OptaTrace with existing work on differentially private location trace synthesis. Our comparison includes three competitors total:

- OptaTrace is the system proposed in this paper. When we report a certain type of error for OptaTrace, we assume that the synthesis is optimized for that error metric, e.g., when reporting Query Error for OptaTrace we assume Err = Query Error.
- AdaTrace is a state-of-the-art differentially private location trace synthesis system described in [11]. Results for AdaTrace are reported using the parameter and budget settings used in [11].
- EQW is a naive version of OptaTrace in which no optimization is performed and OptaTrace is executed with fixed equal weights of $w_1 = w_2 = w_3 = w_4$. EQW is included in the comparison to demonstrate the benefit of OptaTrace's optimization module.

Evaluation Metrics: We use four error metrics in optimization and utility loss measurement: *Query Error*, *Pattern Mining Support Error*, *Trip Error*, and *Travel Distance Error*. According to the utility categories listed in the utility module (Section III-B), Query Error falls under the category of Spatial Density and Locality Metrics; Trip Error and Travel Distance Error fall under the category of Spatio-Temporal Travel Metrics; and Pattern Mining Support Error falls under the category of Pattern Mining Metrics.

Query Error is a popular measure for evaluating noisy data quality. Consider spatial counting queries of the form:

TABLE I: Comparing our proposed OptaTrace system against AdaTrace and EQW. Results across four error metrics, three ε values and three datasets agree that OptaTrace provides higher utility (lower error) compared to AdaTrace and EQW.

		Taxi			**Brinkhoff-4k**			**Brinkhoff-20k**		
		EQW	AdaTrace	OptaTrace	EQW	AdaTrace	OptaTrace	EQW	AdaTrace	OptaTrace
Query Error	$\varepsilon = 0.5$	0.095	0.094	0.059	0.204	0.186	0.133	0.151	0.149	0.132
	$\varepsilon = 1.0$	0.082	0.087	0.052	0.210	0.168	0.101	0.123	0.115	0.100
	$\varepsilon = 2.0$	0.091	0.095	0.045	0.128	0.115	0.093	0.132	0.128	0.097
Pat. Min. Sup. Error	$\varepsilon = 0.5$	0.509	0.481	0.418	0.571	0.549	0.518	0.491	0.480	0.458
	$\varepsilon = 1.0$	0.460	0.429	0.359	0.503	0.474	0.445	0.419	0.408	0.375
	$\varepsilon = 2.0$	0.391	0.378	0.339	0.465	0.485	0.423	0.414	0.407	0.379
Trip Error	$\varepsilon = 0.5$	0.151	0.138	0.093	0.257	0.206	0.106	0.075	0.059	0.033
	$\varepsilon = 1.0$	0.096	0.074	0.027	0.162	0.097	0.058	0.036	0.019	0.011
	$\varepsilon = 2.0$	0.025	0.019	0.009	0.098	0.076	0.035	0.018	0.015	0.008
Travel Distance Error	$\varepsilon = 0.5$	0.038	0.038	0.025	0.099	0.091	0.069	0.063	0.064	0.055
	$\varepsilon = 1.0$	0.027	0.022	0.018	0.069	0.060	0.051	0.055	0.052	0.049
	$\varepsilon = 2.0$	0.023	0.021	0.016	0.057	0.049	0.041	0.052	0.049	0.048

"Retrieve the number of traces passing through geographical region X". Let Q denote a query of this form and $Q(D)$ denote its answer when issued on dataset D. The Query Error is:

$$\text{Query Error} = \frac{|Q(D) - Q(D_{syn})|}{max\{Q(D), b\}} \tag{4}$$

where b is a sanity bound to mitigate the effect of extremely selective queries. We set $b = 0.01 \times |D|$. We generate 200 random queries by changing the geographical region of the query and report the average Query Error across all queries.

Pattern Mining Support Error measures the error in the support values of frequent mobility patterns. Let P denote a pattern as an ordered sequence of cells, e.g., $P : C_3 \rightarrow C_5 \rightarrow C_1$. We define the support of a pattern, $supp(D, P)$, as the number of occurrences of P in dataset D. We mine the top-k patterns from the real dataset D, i.e., the k patterns with highest support, denoted by $\mathcal{F}^k_{\mathcal{U}}(D)$. Then, the Pattern Mining Support Error is:

$$\frac{\sum_{P \in \mathcal{F}^k_{\mathcal{U}}(D)} \frac{|supp(D,P) - supp(D_{syn},P)|}{supp(D,P)}}{k} \tag{5}$$

We use $k = 100$, minimum pattern length of 2 and maximum pattern length of 8 in our experiments.

Trip Error measures error in preserving the correlations between trips' start and end regions. Recall from Section III-A that $\mathcal{R}_{D,\mathbb{A}}$ denotes the trip distribution of dataset D given grid $\mathbb{A}$. We compute the trip distribution of the real dataset using a 6x6 uniform grid $\mathcal{U}$ (denoted $\mathcal{R}_{D,\mathcal{U}}$) and the synthetic dataset using the same grid (denoted $\mathcal{R}_{D_{syn},\mathcal{U}}$). The Trip Error is defined as the Jensen-Shannon divergence between the two distributions: $JSD(\mathcal{R}_{D,\mathcal{U}}, \mathcal{R}_{D_{syn},\mathcal{U}})$.

Travel Distance Error measures the aggregate error in trip travel distances (travel lengths). We calculate the total travel distance of a trip by summing the distance between each consecutive location reading in that trip. Upon finding the maximum travel distance from the real dataset D, we quantize travel distances into 20 equal sized buckets: $\{[0, x), [x, 2x), ..., [19x, 20x]\}$, where $20x$ is the longest travel distance present in D. For each bucket, we determine how many trips' total travel distances fall into that bucket, thereby obtaining a histogram of travel distance buckets versus counts of trips in each bucket. Let $\mathcal{N}_D$ and $\mathcal{N}_{D_{syn}}$ denote the histograms extracted from D and D_{syn} respectively. Then, the Travel Distance Error is equal to: $JSD(\mathcal{N}_D, \mathcal{N}_{D_{syn}})$.

B. Comparison with Prior Work

In Table I, we compare OptaTrace with AdaTrace and EQW using four error metrics, three ε values and three datasets. Results show that OptaTrace's optimized trace synthesis approach yields substantially lower error compared to other approaches. OptaTrace's utility improvement is most pronounced with the Query Error and Trip Error metrics, with roughly 50% error reduction in terms of Trip Error on average.

Comparing the results obtained using the Brinkhoff-4k (smaller dataset) versus the Brinkhoff-20k (larger dataset), we observe that OptaTrace beats AdaTrace on both datasets, but the amount of error reduction is different. On the smaller dataset, we observe higher error reduction; whereas on the larger dataset, we observe smaller error reduction. The reason is because in the larger dataset, errors are already relatively lower compared to the smaller dataset. Thus, there is relatively less room for error improvement using Bayesian optimization. As a result, we expect OptaTrace to be useful in reducing error on smaller datasets. We also observe that smaller ε often yields larger error in all three competitors (EQW, AdaTrace, OptaTrace), as expected. This also brings larger room for error reduction using optimization when ε is small. Consequently, the difference between OptaTrace's error and EQW's error is often larger when ε is small (e.g., $\varepsilon = 0.5$). Their net difference decreases when ε is large (e.g., $\varepsilon = 2$).

C. Analysis of Weight Parameters

Having demonstrated the utility improvement of OptaTrace compared to prior work, we now exemplify the need for fresh optimization for each different D, ε and Err metric; rather than using a fixed or pre-defined set of weights across multiple datasets, ε values or Err metrics. In Figure 4, we illustrate the optimized weight values (w_1, w_2, w_3, w_4) found

(a) Brinkhoff-20k $\varepsilon = 2$ (b) Brinkhoff-20k $\varepsilon = 1$ (c) Taxi $\varepsilon = 1$

Fig. 4: Optimized values of the weight parameters w_1, w_2, w_3, w_4 found using Bayesian optimization in three different dataset and ε combinations. We observe that the optimized weight values differ from one Err metric to another, as well as from one dataset-ε combination to another.

using Bayesian optimization for three different dataset and ε combinations. We make three observations. First, comparing Figure 4a and 4b, we observe that under the same dataset and Err metric, the optimized weight values may change depending on the value of ε. Although optimized weight values may be similar for some Err metrics (such as Trip Error), they are significantly different for others (such as Query Error and Travel Distance Error). Second, comparing Figure 4b and 4c, we observe that under the same ε and Err metric, the optimized weight values may also change depending on the dataset, as one can visually observe the differences between the results on Brinkhoff-20k versus Taxi. Finally, Figures 4a, 4b and 4c each individually show that the optimized weight values are different for each different Err metric, when the dataset and ε values are constant.

Combining the above three observations, we validate that the optimized weight values depend individually on all three factors: dataset D, ε value, and Err metric. Changing one or more of these factors may result in substantially different optimization results. Consequently, we conclude that OptaTrace's fresh Bayesian optimization for each new dataset, ε, and Err metric is beneficial in improving utility; rather than re-using weight values that were learned under different conditions.

D. Optimization Process and Convergence

In Figure 5, we provide a sample run of the Bayesian optimization process used in OptaTrace. We initiate optimization with the Brinkhoff-4k dataset, $\varepsilon = 1$ and Err = Travel Distance Error, with 100 iterations of search space exploration and 100 iterations of optimization. We track the values of each of the weight parameters w_1, w_2, w_3, w_4 for the whole 200 iterations. The resulting graph is illustrated in Figure 5.

The graph shows that during the search space exploration phase (first 100 iterations) and the early stages of the Bayesian optimization phase (iterations 100-140), there can be larger variances in weight values, as indicated by the large spikes and drops between consecutive iterations. However, as optimization approaches the later rounds, most of the weight values converge to their optimized values, and their variances become smaller. Particularly, in iterations between 180-200,

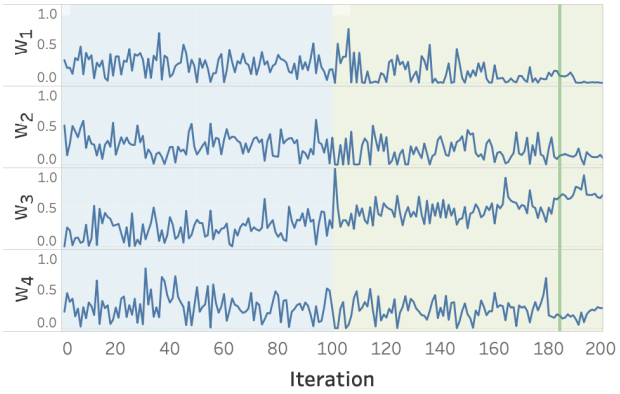


Fig. 5: Values of weight parameters versus iterations of Bayesian optimization (100 iterations of search space exploration + 100 iterations of optimization).

each of the weight values become stable, e.g., w_1 converges to a small non-zero value, w_3 converges to a value closer to 1, and w_2 and w_4 converge to their optimized values between 0 and 0.5. We conclude from this example that reasonably stable convergence can be reached within 200 iterations.

VI. Related Work

Differentially private data synthesis and publication have been active areas of research over the last decade. Several methods were developed for tabular data [27]–[29], set-valued data [30], [31], sequential data [14], transit data [13], and location traces [10]–[12], [32]–[34]. Among them, the methods on differentially private location trace synthesis (DPLTS) are most relevant to our work. In this domain, He et al. developed the DPT system for private trajectory synthesis using hierarchical reference systems, i.e., location discretization with hierarchically organized grids [12]. SGLT system was developed in [35], which synthesizes location traces that satisfy a privacy notion called *plausible deniability*. Plausible deniability relies on a privacy test to reject a candidate synthetic trace from being added to D_{syn} if there are not enough "similar" traces to it in D. It was later shown in [36] that a randomized version of this test can yield a restricted form of (ε, δ)-DP for a certain

set of (ε, δ) parameters. More recently, Gursoy et al. developed DP-Star which was later superseded by the AdaTrace system [10], [11]. In [11], AdaTrace was compared to prior works such as DPT and SGLT, and it was shown that AdaTrace provides superior utility overall. Therefore, we compare our proposed OptaTrace system mainly with AdaTrace.

VII. Conclusion

We presented OptaTrace, an optimization-based approach to differentially private location trace synthesis. Given a real trace dataset D, privacy budget ε and the Err metric, OptaTrace uses Bayesian optimization to minimize $Err(D, D_{syn})$ while ensuring D_{syn} satisfies ε-differential privacy. Compared to prior work, our optimization-based approach is shown to provide substantial error reduction and utility improvement. Contributions of OptaTrace also include a utility module for convenient error measurement and Err metric selection; and a front-end web interface for accessible and easy-to-use DPLTS functionality for non-experts.

Acknowledgments

The authors acknowledge partial support by the National Science Foundation under grants NSF 2038029, NSF 1564097, and an IBM faculty award.

References

[1] Uber movement: Let's find smarter ways forward. movement.uber.com.

[2] (2020) Covid-19 community mobility reports. https://www.google.com/covid19/mobility/.

[3] F. Xu, Z. Tu, Y. Li, P. Zhang, X. Fu, and D. Jin, "Trajectory recovery from ash: User privacy is not preserved in aggregated mobility data," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1241–1250.

[4] A. Pyrgelis, C. Troncoso, and E. De Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," in *Network and Distributed System Security Symposium (NDSS) 2018*, 2018.

[5] C. Y. Ma, D. K. Yau, N. K. Yip, and N. S. Rao, "Privacy vulnerability of published anonymous mobility traces," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 3, pp. 720–733, 2013.

[6] H. Wang, C. Gao, Y. Li, G. Wang, D. Jin, and J. Sun, "De-anonymization of mobility trajectories: Dissecting the gaps between theory and practice," in *The 25th Annual Network & Distributed System Security Symposium (NDSS 18)*, 2018.

[7] S. Chang, C. Li, H. Zhu, T. Lu, and Q. Li, "Revealing privacy vulnerabilities of anonymous trajectories," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 061–12 071, 2018.

[8] E. Kaplan, M. E. Gursoy, M. E. Nergiz, and Y. Saygin, "Location disclosure risks of releasing trajectory distances," *Data & Knowledge Engineering*, vol. 113, pp. 43–63, 2018.

[9] A. Pyrgelis, C. Troncoso, and E. De Cristofaro, "Measuring membership privacy on aggregate location time-series," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*.

[10] M. E. Gursoy, L. Liu, S. Truex, and L. Yu, "Differentially private and utility preserving publication of trajectory data," *IEEE Transactions on Mobile Computing*, vol. 18, no. 10, pp. 2315–2329, 2018.

[11] M. E. Gursoy, L. Liu, S. Truex, L. Yu, and W. Wei, "Utility-aware synthesis of differentially private and attack-resilient location traces," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 196–211.

[12] X. He, G. Cormode, A. Machanavajjhala, C. M. Procopiuc, and D. Srivastava, "Dpt: Differentially private trajectory synthesis using hierarchical reference systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1154–1165, 2015.

[13] R. Chen, B. Fung, B. C. Desai, and N. M. Sossou, "Differentially private transit data publication: a case study on the montreal transportation system," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2012, pp. 213–221.

[14] R. Chen, G. Acs, and C. Castelluccia, "Differentially private sequential data publication via variable-length n-grams," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. ACM, 2012, pp. 638–649.

[15] C. Dwork, "Differential privacy: A survey of results," in *International Conference on Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2013, pp. 901–914.

[18] L. Yu, L. Liu, and C. Pu, "Dynamic differential location privacy with personalized error bounds," in *Network and Distributed System Security Symposium (NDSS '17)*, 2017.

[19] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, "Next place prediction using mobility markov chains," in *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*. ACM, 2012, p. 3.

[20] X. Lu, E. Wetter, N. Bharti, A. J. Tatem, and L. Bengtsson, "Approaching the limit of predictability in human mobility," *Scientific Reports*, vol. 3, p. 2923, 2013.

[21] P. Rathore, D. Kumar, S. Rajasegarar, M. Palaniswami, and J. C. Bezdek, "A scalable framework for trajectory prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3860–3874, 2019.

[22] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[23] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[24] Welcome to py4j – py4j. https://www.py4j.org/.

[25] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi–passenger demand using streaming data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1393–1402, 2013.

[26] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.

[27] N. Mohammed, R. Chen, B. Fung, and P. S. Yu, "Differentially private data release for data mining," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2011, pp. 493–501.

[28] C. Xu, J. Ren, Y. Zhang, Z. Qin, and K. Ren, "Dppro: Differentially private high-dimensional data release via random projection," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 12, pp. 3081–3093, 2017.

[29] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Privbayes: Private data release via bayesian networks," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 4, pp. 1–41, 2017.

[30] R. Chen, N. Mohammed, B. C. Fung, B. C. Desai, and L. Xiong, "Publishing set-valued data via differential privacy," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1087–1098, 2011.

[31] X. Zhang, X. Meng, and R. Chen, "Differentially private set-valued data release against incremental updates," in *International Conference on Database Systems for Advanced Applications*. Springer, 2013, pp. 392–406.

[32] N. Wang and M. S. Kankanhalli, "Protecting sensitive place visits in privacy-preserving trajectory publishing," *Computers & Security*, 2020.

[33] F. Deldar and M. Abadi, "Enhancing spatial and temporal utilities in differentially private moving objects database release," *International Journal of Information Security*, pp. 1–23, 2020.

[34] X. Ding, W. Zhou, S. Sheng, Z. Bao, R. Choo, and H. Jin, "Differentially private publication of streaming trajectory data," *Information Sciences*, 2020.

[35] V. Bindschaedler and R. Shokri, "Synthesizing plausible privacy-preserving location traces," in *2016 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2016, pp. 546–563.

[36] V. Bindschaedler, R. Shokri, and C. A. Gunter, "Plausible deniability for privacy-preserving data synthesis," *Proceedings of the VLDB Endowment*, vol. 10, no. 5, pp. 481–492, 2017.