Differentially Private Synthetic Data Generation Of Data Collected Over Time

By

Girish Kumar
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Thomas Strohmer, Chair

Naoki Saito

Chen-Nee Chuah

Committee in Charge

2024

© Girish Kumar, 2024. All rights reserved.

To my mother and my wife.

Contents

Abstract

Differential privacy has seen a lot of growth in the last decade and has been accepted as a rigorous definition of privacy. A key use case of differential privacy is to generate synthetic data that can be released to the public without concerns about revealing sensitive information. Much of the research in differential privacy has focused on offline applications with the assumption that all data is available at once. When these algorithms are applied in practice to streams where data is collected over time, this either violates the privacy guarantees or results in poor utility.

In this dissertation, we propose three methods that target the task of generating differentially private synthetic data for three different use cases. In Chapters 3 and 4 we look at streams in spatial and tabular spaces but with the assumption that the number of contributions by any particular user is limited. In Chapter 5 we consider the task of generating trajectories that preserve the correlation between points submitted by a user without restricting the number of contributions they make. We show that these proposed methods perform better than existing baselines and have the potential to be adopted for real-world datasets.

Acknowledgments

My PhD program journey has been a challenging, learning, and rewarding experience. I will always cherish these past five years at UC Davis. I am extremely grateful to have had the opportunity to pursue a doctoral degree. First and foremost, I would like to express my sincere gratitude to my advisor, Professor Thomas Strohmer. Thomas introduced me to the world of research and helped me complete the program successfully. Not only was Thomas approachable and patient throughout my PhD journey, but he was also very understanding and supportive of my decisions. I believe he genuinely cares about and works in the best interest of his students. It is impossible to quantify how much I have learned by working with him, and I will be applying these lessons to various aspects of my future endeavors. I would also like to acknowledge the support of the following grants from Thomas: NSF DMS2027248, NSF DMS-2208356, and NIH R01HL1635.

Thomas also exposed me to collaborations outside of the mathematics department at UC Davis. I am extremely thankful to have the opportunity to work with Professor Roman Vershynin from the University of California, Irvine. His guidance and critical discussion throughout our research collaboration has left a deep impact on me. I will be forever motivated to approach any new problem rigorously and yet with simplicity. I am also grateful to have collaborated with Doctor Jason Yeates Adams (UC Davis Medical Group) on multimodal learning and synthetic data generation projects for Healthcare datasets. My learnings about the collection, analysis, and privacy challenges surrounding healthcare datasets have given me a better understanding of real-world challenges.

I am deeply indebted to Professor Naoki Saito and Professor Chen-Nee Chuah for being a valuable part of my qualifying examination and dissertation committee. I am grateful for their feedback and guidance throughout these processes. I am also thankful to Professor Luis Rademacher and Professor Xin Liu for being part of my qualifying examination committee and helping me achieve my research direction.

I feel fortunate to have formed so many wonderful friendships at Davis. Thank you Appilineni Kushal, Shizhou Xu, Xue Feng, Yuan Ni, Alvin Chen, Stefan Broecker, Benjamin Godkin, Satyabrata Sarangi, and many more for all the wonderful discussions on academia and life. I am also thankful to my childhood friends Abhishek Mani, Venkata Gangadhar Kanchu, Mohammed Asjad Quadeer, and Aviral Shukla for their constant friendship and the joyous times we have spent together over the holidays.

My deepest gratitude goes to my mother, who exemplified the highest qualities of humanity and always inspired me to be compassionate, hardworking, and resilient. Though I lost her during the pandemic, her teachings will continue to inspire me throughout my life. I would also like to thank my father and my sisters, Rakhi and Luxmi, for their unconditional love. Finally, my PhD journey would not have been possible without the unwavering support of my loving wife, Kumari Nishu. While I will be the one earning the doctoral degree, she has been with me every step of the way, sharing the burden of all my personal challenges throughout this journey.

CHAPTER 1

Introduction

1.1. The abundance and scarcity of data

Artificial Intelligence (AI) based algorithms have become ubiquitous in today's world. Many corporate and research organizations rely on the availability of sufficient high-quality data to create such algorithms. Indeed with the integration of the internet in almost every aspect of our life, there is an abundance of data being captured from each individual in the digital world. However much of this data cannot simply be shared publicly for research and development purposes as that will likely violate the privacy of individuals.

De-identification techniques that simply remove Personal Identifiable Information (PII) have been shown to be insufficient in protecting the privacy of individuals [21, 26, 30, 39, 42, 60]. Even aggregated data has been shown to be susceptible to adversarial attacks [25, 76]. A prominent technique in some of these adversarial attacks is cross-referencing with other publicly available information. More sophisticated techniques such as k-annonimity, which have been used widely in the past [55, 65], have also been proven to be prone to such attacks [15].

Differential Privacy [22] has emerged as a strong and rigorous method of guaranteeing the privacy of individuals. Not only do we have a mathematically proven guarantee but any post-processing on the output, such as cross-referencing with other datasets, does not violate the privacy guarantees. Thus differential privacy has been used by many notable institutions such as US Census [4], Google [27], and Apple [68]. Differential Privacy has seen a lot of research across multiple applications such as statistical query answering [56], regression [63], clustering [61], and large-scale deep learning [3].

A promising idea to overcome these constraints on data sharing is to create privacy-preserving synthetic data. This research focuses on providing algorithms that can generate synthetic data. The synthetic data produced should have similar properties as the true data for it to be useful. It must also protect the privacy of the individuals who contributed to the true data. Differential

privacy has been used for such synthetic data generation across various data domains such as tabular microdata [36, 44, 57, 67, 78], natural language [52, 77], and images [69, 75].
Differential privacy is parameterized by a constant ε which is known as the *privacy budget*. Compared to another differentially private algorithm, a better algorithm uses less privacy budget and yet generates synthetic data with similar utility. We discuss differential privacy formally in Section 2.1. Intuitively, it offers robustness in the output distribution against a change caused by removing a user's contribution. This robustness implies that the output of a differentially private algorithm does not depend too much on data from any particular user, which in turn implies privacy. To achieve this robustness, the algorithm spends some of the privacy budget ε and typically adds white noise to mask the change that may occur by removing a user's data. This change is referred to as the sensitivity of the algorithm.

1.2. Problem scope and prior work

1.2.1. Data collected over time. In this dissertation, we are interested in generating synthetic data when the true data has a time dimension. Some examples of such datasets are hospitalization, recovery, and release records of patients; a dataset captured in a multi-year study to understand the effect of a public policy over time; and GPS trajectories for taxis in a city. As compared to datasets which do not have a time dimension, there are several key challenges here,

(1) *Change in data distribution over time:* Since the data has a time dimension, the underlying distribution of the private data may change significantly over time. To some extent, this problem can be solved by considering the time when the data was recorded as simply another attribute of the dataset. However other challenges as we mention below will persist.
(2) *Correlation between data contributions by a user over time:* In most algorithms in differential privacy, there is an assumption that a user can contribute only one data point. Even when the user is allowed to contribute more than one point, those contributions are considered independent of one another. These assumptions are invalid for certain datasets such as a collection of GPS trajectories. A user may start from a certain origin and move along a pre-defined path to reach a certain destination. In this case, the points within the trajectory are certainly correlated.

(3) *Large sensitivity:* The amount of noise required to provide robustness against a change of point is typically a function of the sensitivity of the algorithm. In datasets collected over time, removing a point at some time will likely affect measurements on the datasets at all future times as well. Thus resulting in a large sensitivity that scales with time, which in turn would require a large noise to be added to achieve differential privacy.

(4) *Building a streaming algorithm:* Since the dataset can be collected over a large time horizon, the curator may wish to release the synthetic data as the true data is being collected. This requires streaming algorithms for synthetic data generation which, as we will discuss in Section 1.2.2, are very different from one-time release algorithms.

In this dissertation, we will provide algorithms for both one-time release and streaming release of synthetic data with time dimension. Due to the challenges mentioned above, it is difficult to give an algorithm that is practical and can work without any constraints on the data model. Hence, we first look at the streaming release of privacy-preserving synthetic data for low and high dimensional data in Chapters 3 and 4 respectively. In these works, we assume that the number of times a user can contribute to the stream is limited.

It is particularly challenging to construct an algorithm that does not require limiting the number of user contributions over time. We look at one such case in Chapter 5 with constraints that we consider low-dimensional data and one-time release. Note that, with these constraints, the dataset can be regarded as a collection of spatial trajectories. To develop a streaming algorithm for high-dimensional data, with unlimited contributions by a user over time, remains an open problem. We discuss more about this in Chapter 6.

1.2.2. One-time release vs streaming algorithms.

Despite a vast body of research, the majority of developments in differential privacy have been restricted to a one-time collection or release of information. In many practical applications, the techniques are required to be applied either on an event basis or a regular interval. Many industry applications of differential privacy simply re-run the algorithm on all data collected so far, thus making the naive (and usually, incorrect) assumption that future data contributions by users are completely independent of the past. This either violates the privacy guarantee completely or results in a very superficial guarantee of privacy [66].

Differential privacy can also be used for streaming data allowing the privacy-preserving release of information in an online manner with a guarantee that spans over the entire time horizon [23]. However, most existing algorithms using this concept such as [12, 19, 23, 45] are limited to the release of a statistic after observing a stream of one-dimensional input (typically a bit stream) and have not been explored for tasks such as synthetic data generation. Similar to these researches, we use a notion of differential privacy that accepts a stream as input and guarantees privacy over the entire time horizon, sometimes referred to as Continual differential privacy. In the differential privacy literature, one-time and streaming algorithms are also referred to as offline and online algorithms respectively.

Early works such as [23] and [12] explores the release of bit count over an infinite stream of bits and introduce various effective algorithms, including the Binary Tree mechanism. We refer to these mechanisms here as *Counters* and use them as a subroutine of our algorithm. In [24], the authors build upon the work in [23] and use the Binary Tree mechanism together with an online partitioning algorithm to answer range queries. However, the problem considered is answering queries on offline datasets. In [13, 71] authors further build upon the task of privately releasing a stream of bits or integers under user and event-level privacy. A recent work [40] addresses deletion when observing a stream under differential privacy. They approach the problem of releasing a count of distinct elements in a stream. Since these works approach the task of counting a stream of one-dimensional data, they are different from our use case of multi-dimensional density estimation and synthetic data generation. The Binary Tree Mechanism has also been used in other problems such as online convex learning [32] and deep learning [46], under the name *tree-based aggregation trick*. Many works with streaming input have also explored the use of local differential privacy for the collection and release of time series data such as telemetry [19, 45].

To the best of our knowledge two very recent works [10] and [38] are the only others to approach the task of releasing a synthetic stream with differential privacy. However [10] approach a specific problem where the universe consists of a fixed set of users, each contributing to the dataset at all times. Moreover, a user's contribution is limited to one bit at a time, and the generated synthetic data is derived to answer a fixed set of queries. In contrast, we will allow multi-dimensional continuous value input from an arbitrary number of users and demonstrate the utility over randomly generated queries. The authors in [38] on the other hand provide an algorithm with theoretical guarantees but do not provide any experimental evaluations of the algorithm. They use an algorithm based on the hierarchical decomposition of the space, and as we will discuss in detail in Section 1, such algorithms do not scale well in practice for high-dimensional data.

1.2.3. Low-dimensional streams (Chapter 3). A key example of a low-dimensional dataset is the collection of spatial (latitude and longitude) coordinates. Many data-driven applications require frequent access to the user's location to offer their services more efficiently. These are often called Location Based Services (LBS). Examples of LBS include queries for nearby businesses [7], calling for taxi pickup [50, 80], and local weather information [74].

Much of location data contains sensitive information about a user in itself or when cross-referenced with additional information. Several studies [42] have shown that publishing location data is susceptible to revealing sensitive details about the user and simply de-identification [21] or aggregation [76] is not sufficient. Thus, privacy concerns limit the usage and distribution of sensitive user data.

Differential privacy has also been explored extensively for various use cases concerning spatial datasets such as answering range queries [79], collecting user location data [48], and collecting user trajectories [51]. The methods can be broadly classified into two categories based on whether or not the curator is a trusted entity. If the curator is not-trusted, more strict privacy guarantees such as Local Differential Privacy (LDP) and Geo-Indistinguishability are used and action is taken at the user level before the data reaches the server. We focus on the case where data is stored in a trusted server and the curator has access to the true data of all users. This setup is more suited for publishing privacy-preserving microdata and aggregate statistics.

In particular for synthetic spatial microdata generation, a popular approach is to learn the density of true data as a histogram over a Private Spatial Decomposition (PSD) of the data domain and sample from this histogram [28, 47, 62, 79]. Our work also uses this approach and builds upon the method PrivTree [79] which is a very effective algorithm for generating PSD. We refer the reader to [48] for a survey of some other related methods. We are however interested in releasing synthetic spatial streams and these offline algorithms assume we have access to the entire dataset at once so they do not apply directly to our streaming use case.

1.2.4. High-dimensional streams (Chapter 4). Let us first talk about offline algorithms for high-dimensional datasets. An example of a high-dimensional dataset is demographic data for the census. Such a dataset is also sometimes termed as *microdata*. Hierarchical decomposition based algorithms that are generally used for low-dimensional data do not scale well in high-dimensions. A popular approach in the high-dimensional setting is to create a synthetic dataset that agrees with the true private data on some low-dimensional statistics. Typically these low-dimensional statistics are marginal queries.

A marginal query counts the number of instances in the dataset where a subset of columns takes a given combination of values. For example, suppose we have a census-like dataset where some of the demographics are age, marital status, and income. An example marginal query on these columns is - how many individuals in the dataset are age 30, never married, and have income more than $\$100,000$ per annum. Given a set of such queries, a good synthetic dataset is one on which the result of these queries is close to the true values in some pre-defined measure. Algorithms that use this method typically aim to preserve all 2-way or 3-way or both marginals of the dataset. This method has been explored in various previous works [6,36,54,57,78]. A straightforward approach when using marginal queries will be to (1) measure all queries (that are required to be preserved) on true data; (2) add noise to them to achieve differential privacy; and (3) generate synthetic data that complies with these noisy measurements.

Many algorithms have been proposed to generate synthetic data that complies with marginals (step 3 above) such as the Multiplicative Weights (MW) algorithm [36], the Relaxed Adaptive Projection (RAP) algorithm [6], and the PrivBayes algorithm [78]. A key difference among these algorithms is their assumption of how underlying data distribution is modeled using the marginal queries. The MW algorithm directly maintains an estimated distribution on the entire data space and adjusts the distribution to comply with the marginal queries. The algorithm thus quickly becomes intractable for reasonably high dimensions of data. The RAP algorithm relaxes the entire space to a continuous space and does gradient descent-based optimization on this space to find the data distribution. The PrivBayes algorithm uses a subset of the marginals and models the distribution as a Bayesian network. Both the RAP and the PrivBayes algorithm typically scale well with high-dimensional data.

In our analysis, all of these methods are outperformed by the Probabilistic Graphical Model (PGM) technique [57] to model the data distribution based on marginals. As the name suggests, PGM creates a graphical model based on a subset of the marginals and uses their measurements to approximate the data distribution.

However, the three-step simple approach mentioned earlier is inefficient since it does not take advantage of the correlation among queries. Moreover, for high-dimensional data, measuring all queries results in less privacy budget available for each of the queries. To overcome these issues, many prior works, such as [6,36,57,78], use the *select, measure, learn, and iterate* paradigm. In this approach, the algorithm iteratively selects a query (typically the worst-performing query), measures it, and uses only the set of noisy measurements so far for generating synthetic data. An early work that used this paradigm is MWEM [36] which combined the Multiplicative Weights algorithm with the Exponential Mechanism (a differentially private selection algorithm) to generate synthetic data. MWEM is typically the best-performing algorithm if it can scale to the data dimensionality [54]. In this work, we use MWEM+PGM [57], a scalable version of the MWEM approach which replaces modeling the data distribution from MW to PrivatePGM.

However, similar to the case of low-dimensional data, these algorithms are for offline dataset release and do not directly apply to our use case of privacy-preserving streaming of synthetic data.

1.2.5. Trajectories (Chapter 5). In Chapters 4 and 3, we do not model the dynamics of any individual user's data over time but rather the dynamics of the entire dataset. This is justified with the assumption that a user contributes only a few points throughout time (low sensitivity) and we have discussed some applications where such an assumption is justified. In other applications, the assumption of low sensitivity is not justified. A prominent example is *trajectories* containing spatial coordinates of a moving body, such as an individual [16], a taxi [70], or a ship [41]. Thus, in this case, we not only intend to preserve the aggregated information in the dataset but also the correlation among points contributed by an individual user. However, to reduce the complexity we consider a one-time release, instead of a streaming release of data.

Many previous works have considered the task of generating privacy-preserving synthetic trajectory datasets. The majority of the works can be divided into two classes based on whether or not they satisfy Local differential privacy (LDP). We briefly mentioned LDP in Section 1.2.3 and will define it more rigorously in Section 2.5. In this work, we do not consider LDP, instead, we assume that the curator can be trusted and use (central) differential privacy.

Many existing works, such as [33, 37, 64, 70], have focused on producing a privacy-preserving synthetic trajectory dataset. To model the correlation among the points within the trajectory, these works assume that the trajectory is generated as a Markov chain. With this assumption, they model the generation of the next point in any trajectory as a transition from one state of a Markov process to another. The transition probabilities are estimated while preserving differential privacy. In order to reduce the complexity of the algorithm, a key step used in these works is *discretization*. During discretization, the space of points is partitioned, and then individual points are mapped to the subset in the partition they belong to. Thus, the number of unique states for the Markov Process decreases dramatically.

Despite a large body of work, the problem of generating realistic trajectories with high utility remains unsolved. We believe this is largely because existing methods will not scale well with a fine discretization resolution. In Chapter 5 we first discuss some of the challenges that the existing methods will face when dealing with a high-resolution discretization. Later, we provide a method that will address these challenges.

1.3. Contributions

In Chapters 3 and 4, we introduce the novel task of privacy-preserving synthetic multi-dimensional stream generation. Chapter 3 draws from the work written by the dissertation author as the first author on differentially private synthetic spatial stream generation [49]. In these chapters, we are particularly interested in low-sensitivity datasets collected over time. For motivation, consider the publication of the coordinates (latitude and longitude) of residential locations of people with active coronavirus infection. As people get infected or recover, the dataset evolves and the density of infection across our domain can dynamically change over time. Such a dataset can be extremely helpful in making public policy and health decisions by tracking the spread of a pandemic over both time and space. This dataset has low sensitivity in the sense that it is rare for a person to get infected more than a few times in, say, a year. This allows us to limit the number of contributions a user can make in the stream over time.

In these chapters, we present methods that can be used to generate synthetic data streams with differential privacy and we demonstrate its utility on real-world datasets. Our contributions are summarized below.

(1) To the best of our knowledge, we present the first differentially private streaming algorithm for the release of multi-dimensional synthetic data;

(2) we present a meta-framework that can be applied to a large number of counting algorithms to resume differentially private counting on regular intervals;

(3) we further demonstrate the utility of our algorithms for synthetic data generation on both simulated and real-world spatial datasets;

(4) furthermore, our algorithm can handle both the addition and the deletion of data points (sometimes referred to as turnstile model), and thus dynamic changes of a dataset over time.

In Chapter 5, we discuss recent works that have addressed the problem of generating a privacy-preserving synthetic trajectory dataset. We also discuss the limitations of these methods and why they will not have good utility when scaled to a high-resolution grid for discretization.

We then provide our method that accommodates any spatial constraints and works for many general spaces $\mathcal{X}$ such as the natural earth landscape of a country, and a graph representing the road network of a city. We use a density-based discretization approach that produces partitions of very high resolution in dense areas to efficiently capture the spatial information in the dataset. We also provide experiments that highlight the problems discussed and validate our proposed method.

CHAPTER 2

Background on Differential Privacy

2.1. A rigorous definition of privacy

Let us suppose we are doing a study whose results will be publicly available. The study requires the participants to share some information that is considered private by them. As an example, consider the study to find a correlation between smoking and the severity of illness due to COVID-19 and for the sake of argument let us consider that there is a strong correlation. The idea behind differential privacy is that this correlation should still be present and almost as strong even if a particular participant decides not to participate in the study. Then, even if the study results are made publicly available, differential privacy provides an individual with plausible deniability for their participation in the study. To rigorously define this concept we need a notion of neighboring datasets.

2.1.1. Neighboring datasets. Let $\mathcal{X}$ be a space of data points. Let U be a finite set of users. Let $f : U \times \mathcal{X} \to \mathbb{N}$ be a dataset represented as a function such that $f(u, x)$ denotes the number of times a user $u \in U$ contributed the point $x \in \mathcal{X}$ in the dataset. Here, and everywhere else in this work, we abuse the notation and denote with $\mathbb{N}$ the set of all natural numbers including the digit 0. Then, for any user $u \in U$, a neighboring dataset, denoted as $\tilde{f}_u$, can be created by removing all contributions by the user u from f. That is, (1) $\tilde{f}_u(v, x) = f(v, x)$ for all $x \in \mathcal{X}$ and $v \in U \setminus \{u\}$; and (2) $\tilde{f}_u(u, x) = 0$ for all $x \in \mathcal{X}$.

In this work, we first build algorithms with the assumption that a user contributes at most one data point. Later, using composition properties of differential privacy (Section 2.1.4.1), we extend the algorithm to the case where a user may contribute multiple data points. Hence, we relax the notation of a dataset to $f : \mathcal{X} \to \mathbb{N}$ such that $f(x)$ denotes the number of times $x \in \mathcal{X}$ appears in the dataset. The resulting definition of neighboring datasets is given in Definition 2.1.1. The notation $\mathbb{N}^{\mathcal{X}}$ denotes the power set containing all possible functions of the form $f : \mathcal{X} \to \mathbb{N}$.

Definition 2.1.1 (Neighboring Datasets). *Two datasets $f, \tilde{f} : \mathcal{X} \to \mathbb{N}$ are said to be neighbors if they differ in the count of exactly one point by a unit, that is, there exists $x \in \mathcal{X}$ such that $\left|f(x) - \tilde{f}(x)\right| = 1$ for some $x \in \mathcal{X}$ and $\tilde{f}(y) = f(y)$ for all $y \in \mathcal{X} \setminus \{x\}$.*

2.1.2. Definition. We can now define Differential privacy using the concept of neighboring datasets.

Definition 2.1.2 (Differential Privacy). *A randomized algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}$ is said to satisfy ε-differential privacy if for all neighboring datasets f and $\tilde{f}$ and for any measurable $S \subseteq \mathcal{R}$ we have,*

$$\mathbb{P}\left\{\mathcal{A}(f) \in S\right\} \leq e^{\varepsilon} \cdot \mathbb{P}\left\{\mathcal{A}(\tilde{f}) \in S\right\},$$

where the probability is taken over the randomness of $\mathcal{A}$.

Intuitively, differential privacy guarantees a form of robustness - the probability of observing the output of a randomized algorithm $\mathcal{A}$ only differs by a "small" multiplicative factor of e^{ε} if the input dataset differs in the data from an individual. Note that differential privacy is a property of the algorithm $\mathcal{A}$ and not the output $\mathcal{A}(f)$.

2.1.3. The privacy budget ε and the privacy-utility trade-off. The parameter $\varepsilon \geq 0$ is often referred to as the *privacy budget* and acts as a way to control the privacy and utility trade-off. To understand this trade-off let us consider two extreme algorithms - (1) $\mathcal{A}(f) = \emptyset$ and (2) $\mathcal{A}(f) = f$, for any input f. In case (1), algorithm $\mathcal{A}$ can be shown to uphold Definition 2.1.2 for $\varepsilon = 0$ however, since the output is always an empty set the algorithm has no utility. On the other extreme, in case (2), the algorithm has perfect utility as it outputs the entire dataset but it does not satisfy differential privacy for any non-zero real value of ε. The name *budget* has another relevance due to the composition properties of Differential Privacy and we discuss this connection in Section 2.1.4.1.

2.1.4. Properties of differential privacy.

2.1.4.1. *Composition.* To design complex algorithms, we often rely on smaller sub-routines that compose together to form complex logic. The definition of differential privacy satisfies composition properties allowing us to break complex tasks into smaller subroutines that satisfy differential privacy. Theorem 2.1.3 and 2.1.4 are two key composition properties of differential privacy. Sequential

Composition allows us to divide the privacy parameter ε into smaller parameters for subroutine and get multiple outputs from the same data. Thus the parameter ε is sometimes referred to as the privacy budget as each subroutine may take some part of it. Whereas in parallel composition, the data itself gets divided into disjoint parts, of which at most one part may see a change in a neighboring dataset. Thus the privacy budget is only consumed by at most one subroutine. We present these concepts more rigorously below.

THEOREM 2.1.3 (Sequential composition). *Let $\mathcal{A}_1 : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_1$ and $\mathcal{A}_2 : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_2$ be randomized algorithms that satisfy ε_1 and ε_2-differential privacy respectively. Then, an algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_1 \times \mathcal{R}_2$ defined as $\mathcal{A}(f) := \big(\mathcal{A}_1(f), \mathcal{A}_2(f)\big)$, for any input dataset f, satisfies $(\varepsilon_1 + \varepsilon_2)$-differential privacy.*

THEOREM 2.1.4 (Parallel composition). *Let $\{\mathcal{X}_1, \mathcal{X}_2\}$ be an arbitrary disjoint partition of the space $\mathcal{X}$. Let $\mathcal{A}_1 : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_1$ and $\mathcal{A}_2 : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_2$ be randomized algorithms that satisfy ε_1 and ε_2-differential privacy respectively. Let, for any $i \in \{1, 2\}$, $f|_{\mathcal{X}_i}$ denote the restriction of f on $\mathcal{X}_i$ such that*

$$f|_{\mathcal{X}_i}(x) = \begin{cases} f(x), & x \in \mathcal{X}_i, \\ 0, & x \in \mathcal{X} \setminus \mathcal{X}_i. \end{cases}$$

Then, an algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_1 \times \mathcal{R}_2$ defined as $\mathcal{A}(f) := \Big(\mathcal{A}_1\left(f|_{\mathcal{X}_1}\right), \mathcal{A}_2\left(f|_{\mathcal{X}_2}\right)\Big)$ satisfies $\max(\varepsilon_1, \varepsilon_2)$-differential privacy.

2.1.5. Post-processing. The post-processing guarantee of differential privacy (Theorem 2.1.5) is very desirable. It implies that no matter what happens with the output of a differentially private algorithm, the privacy guarantees that the algorithm satisfies remain unaffected.

THEOREM 2.1.5 (Post-processing gurantees). *Let $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_1$ be a randomized algorithm that satisfies ε-differential privacy. Let $\mathcal{M} : \mathcal{R}_1 \to \mathcal{R}_2$ be an arbitrary randomized mapping. Then a mapping $\mathcal{M} \circ \mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}_2$ defined as $(\mathcal{M} \circ \mathcal{A})(f) := \mathcal{M}\left(\mathcal{A}(f)\right)$ for any dataset $f \in \mathbb{N}^{\mathcal{X}}$ satisfies ε-differential privacy.*

2.2. How to achieve differential privacy?

In most cases, we achieve differential privacy by adding noise sampled from a distribution. However, naively adding noise may result in a poor utility of the algorithm. In sections 2.2.1 and 2.2.2, we present two well-known differentially private algorithms that we will use as subroutines of our more complex algorithms.

2.2.1. Measurement. A key use of differential privacy is to find answers to some statistical queries on a private dataset. As a simple motivating example, suppose we want to find the mean age using a dataset that contains the age of individuals. The Laplace Mechanism is the most fundamental algorithm in differential privacy for measuring such statistics about a given dataset. We state the mechanism in Definition 2.2.3. The mechanism gets its name since the noise added is drawn from the Laplace distribution (Definition 2.2.1).

2.2.1.1. *Laplace distribution and related properties.* In this section, we first present a definition of the Laplace distribution, followed by some bounds on the concentration of Laplace random variables.

Definition 2.2.1 (Laplace distribution). *A random variable X is said to be drawn from the Laplace distribution* $\mathrm{Lap}(\mu, b)$ *if its probability density function follows*

$$f(X \mid \mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right), \tag{2.1}$$

where μ and b are the location and scale parameters respectively. Note that $\mathbb{E}[X] = \mu$ *and* $\mathrm{Var}(X) = 2b^2$. *We simply use the notation* $\mathrm{Lap}(b)$ *if the location parameter* $\mu = 0$.

The following lemmas about the concentration of the Laplace distribution will be useful for our analysis later.

Lemma 2.2.1.1 (Concentration of a Laplace random variable). *Let $X \sim Lap(b)$. Then for any $\beta > 0$ we have,*

$$\mathbb{P}\left\{|X| > b\log\frac{1}{\beta}\right\} = \beta. \tag{2.2}$$

Proof. Simply by the probability density function we have,

$$\mathbb{P}\left\{|X| > \alpha\right\} = 2\int_{\alpha}^{\infty}\frac{1}{2b}\exp\left(-\frac{x}{b}\right) = \exp\left(-\frac{\alpha}{b}\right).$$

We get the desired result by using $\beta = \exp\left(-\frac{\alpha}{b}\right)$. ☐

2.2.1.2. *Laplace Mechanism.* As mentioned previously, we will use the Laplace mechanism to answer statistical queries about the dataset. Such a query is usually represented as a function $q : \mathbb{N}^{\mathcal{X}} \to \mathbb{R}^d$. For example, let $\mathcal{X} = \{0, 1, \ldots, 99\}$ and $f(x)$ denote the number of participants of age $x \in \mathcal{X}$ present in the dataset f. Let $q : \mathbb{N}^{\mathcal{X}} \to \mathbb{R}^{10}$ be a query defined as

$$q_i(f) \coloneqq \sum_{x \in \left\{10(i-1), \ldots, 10i-1\right\}} f(x),$$

that is $q_i(f)$ denotes the number of individuals in the dataset f with age in the range $[10(i-1), 10i-1)$. To define the Laplace mechanism, we need a notion of sensitivity. Intuitively, it represents the amount of maximum change that can occur in the query output when replacing an input dataset with a neighboring dataset.

Definition 2.2.2 (ℓ_1 sensitivity). *The ℓ_1 sensitivity of a function $q : \mathbb{N}^{\mathcal{X}} \to \mathbb{R}^d$, denoted as Δ_q, is defined as*

$$\Delta_q \coloneqq \max_{\substack{f, \tilde{f} \in \mathbb{N}^{\mathcal{X}}, \\ \left\| f - \tilde{f} \right\| = 1}} \left\| q(f) - q(\tilde{f}) \right\|_1 . \tag{2.3}$$

Definition 2.2.3 (Laplace Mechanism). *The Laplace mechanism is defined as an algorithm $\mathcal{A}$ such that*

$$\mathcal{A}(f) \coloneqq q(f) + \eta, \tag{2.4}$$

where $\eta \in \mathbb{R}^d$ is a d-dimensional vector such that η_i are i.i.d. random variables for all $i = 1, 2, \ldots, d$ that are drawn from the distribution $\operatorname{Lap}\left(\frac{\Delta_q}{\varepsilon}\right)$.

Theorem 2.2.4 (Privacy of Laplace mechanism). *The Laplace mechanism, as described in Definition 2.2.3, satisfies ε-differential privacy.*

Proof. Let $f, \tilde{f} \in \mathbb{N}^{\mathcal{X}}$ be neighboring datasets such that $\left\| f - \tilde{f} \right\| = 1$. Then for any output $y \in \mathbb{R}^d$,

$$\frac{\mathbb{P}\left\{\mathcal{A}(f) = y\right\}}{\mathbb{P}\left\{\mathcal{A}(\tilde{f}) = y\right\}} = \frac{\mathbb{P}\left\{\eta = y - q(f)\right\}}{\mathbb{P}\left\{\tilde{\eta} = y - q(\tilde{f})\right\}} = \frac{\prod_{i=1}^{d}\left(\frac{\varepsilon}{2\Delta_q}\exp\left(-\frac{\varepsilon\left|(y-q(f))_i\right|}{\Delta_q}\right)\right)}{\prod_{i=1}^{d}\left(\frac{\varepsilon}{2\Delta_q}\exp\left(-\frac{\varepsilon\left|(y-q(\tilde{f}))_i\right|}{\Delta_q}\right)\right)}$$

$$\leq \exp\left(\frac{\varepsilon}{\Delta_q}\left\| q(f) - q(\tilde{f})\right\|_1\right) \leq e^{\varepsilon}$$

□

To prove the utility of the Laplace mechanism we will be using the following Lemma 2.2.4.1 about the concentration of the maximum value of a set of random variables. The proof of Lemma 2.2.4.1 is a simple application of union bound.

Lemma 2.2.4.1 (Concentration of maximum value of random variables). *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that for any $\beta > 0$ there exists $\alpha > 0$ such that $\mathbb{P}\left\{|X_i| > \alpha\right\} \leq \beta$ for each $i = 1, 2, \ldots, n$. Then, for any $\beta > 0$,*

$$\mathbb{P}\left\{\max_{i=1,2,\ldots,n} |X_i| > \alpha\right\} \leq \beta n.$$

Theorem 2.2.5 (Utility of Laplace mechanism). *For any $\beta > 0$, the Laplace mechanism, as described in Definition 2.2.3, satisfies*

$$\mathbb{P}\left\{\left\|\mathcal{A}(f) - q(f)\right\|_\infty \geq \frac{\Delta_q}{\varepsilon}\log\left(\frac{d}{\beta}\right)\right\} \leq \beta. \tag{2.5}$$

Proof. By Lemma 2.2.1.1 we have that for any $\beta > 0$ and for each η_i individually,

$$\mathbb{P}\left\{|\eta_i| > \frac{\Delta_q}{\varepsilon}\log\left(\frac{1}{\beta}\right)\right\} \leq \beta.$$

Hence, by applying Lemma 2.2.4.1 and replacing β with β/d we have the required result. □

2.2.1.3. *Histogram query.* An important case when the Laplace Mechanism is highly effective is when answering a histogram query as defined below.

Definition 2.2.6 (Histogram query). *A function $q : \mathcal{X} \to \{0,1\}^d$ is called a histogram query if it follows that $\left\|q(x)\right\|_1 = 1$ for any $x \in \mathcal{X}$. With a slight abuse of notation, we define an extension of this query to datasets as $q : \mathbb{N}^{\mathcal{X}} \to \mathbb{N}^d$ with*

$$q(f) \coloneqq \sum_{x \in \mathcal{X}} f(x) \cdot q(x), \tag{2.6}$$

for all $f \in \mathbb{N}^{\mathcal{X}}$.

We use the term histogram since if we consider each of the d dimensions of the output as a bin then a histogram query $q : \mathcal{X} \to \{0,1\}^d$ classifies points in $\mathcal{X}$ to one of the d bins and thus its extension $q : \mathbb{N}^{\mathcal{X}} \to \mathbb{N}^d$ counts the number of points in the dataset in each of the d bins. The example query mentioned in Section 2.2.1.2 is indeed a histogram query. Note that the sensitivity of a histogram query over datasets, as defined in Equation 2.3, is 1 and is thus independent of the dimension of query output d. Thus as per Theorem 2.2.5 the accuracy of a histogram query is $\mathcal{O}(\log d)$.

2.2.2. Selection. Let $\mathcal{R}$ be a finite set. Let $u : \mathbb{N}^{\mathcal{X}} \times \mathcal{R} \to \mathbb{R}$ be a function such that $u(f, r)$ denotes the utility of an element $r \in \mathcal{R}$ for a dataset $f \in \mathbb{N}^{\mathcal{X}}$. Our task is to find an element in $\mathcal{R}$ with maximum utility while preserving differential privacy. Note that the term utility is very general and its exact definition is governed by the problem. As an example, suppose we want to find a mode of a given dataset $f \in \mathbb{N}^{\mathcal{X}}$. The mode is any point $x_* \in \mathcal{X}$ such that $x_* = \arg\max_{x \in \mathcal{X}} f(x)$. We can use the exponential mechanism in this case with the utility being the absolute difference between the frequency of any point from the maximum possible frequency in f. Thus in this case $\mathcal{R} = \mathcal{X}$, and $u(f, x; x_*) = \left|f(x) - f(x_*)\right|$ for all $x \in \mathcal{X}$.

Definition 2.2.7 (Exponential mechanism). *Let Δ_u denote the sensitivity of the utility function defined as,*

$$\Delta_u \coloneqq \max_{\substack{f, \tilde{f} \in \mathbb{N}^{\mathcal{X}}, \\ \left\|f - \tilde{f}\right\| = 1}} \max_{r \in \mathcal{R}} \left|u(f, r) - u(\tilde{f}, r)\right|. \tag{2.7}$$

Then, the exponential mechanism is defined as the algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}$ such that, for all $r \in \mathcal{R}$,

$$\mathbb{P}\left\{\mathcal{A}(f) = r\right\} \propto \exp\left(-\frac{\varepsilon}{2\Delta_u} u(f, r)\right).$$

Theorem 2.2.8 (Privacy of exponential mechanism). *The exponential mechanism, as defined in Definition 2.2.7, satisfies ε-differential privacy.*

Proof. Let $f, \tilde{f} \in \mathbb{N}^{\mathcal{X}}$ be neighboring datasets such that $\left\| f - \tilde{f} \right\| = 1$. Then for any output $r \in \mathcal{R}$,

$$\frac{\mathbb{P}\left\{\mathcal{A}(f) = r\right\}}{\mathbb{P}\left\{\mathcal{A}(\tilde{f}) = r\right\}} = \frac{\exp\left(-\frac{\varepsilon}{2\Delta_u} u(f,r)\right)}{\exp\left(-\frac{\varepsilon}{2\Delta_u} u(\tilde{f},r)\right)} \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u} u(\tilde{f},r')\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u} u(f,r')\right)}$$

$$= \exp\left(-\frac{\varepsilon}{2\Delta_u}\left(u(f,r) - u(\tilde{f},r)\right)\right) \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u}\left(u(\tilde{f},r') - u(f,r') + u(f,r')\right)\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u} u(f,r')\right)}$$

$$\leq e^{\varepsilon/2} \cdot e^{\varepsilon/2} \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u} u(f,r')\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\varepsilon}{2\Delta_u} u(f,r')\right)} = e^{\varepsilon}$$

□

Theorem 2.2.9 (Accuracy of exponential mechanism). *For any $\beta > 0$, the exponential mechanism, as defined in Definition 2.2.7, satisfies*

$$\mathbb{P}\left\{u(f, \mathcal{A}(f)) \leq u_{OPT} - \frac{2\Delta_u}{\varepsilon} \ln\left(\frac{|\mathcal{R}|}{\beta}\right)\right\} \leq \beta,$$

where $u_{OPT} = \max_{r \in \mathcal{R}} u(f,r)$ denotes the maximum possible utility.

Proof. Let $\mathcal{R}^* = \{r | u(f,r) = u_{OPT}\}$ be the possible values in $\mathcal{R}$ for which we have the maximum possible utility.

$$\mathbb{P}\left\{\mathcal{A}(f) \leq u_{OPT} - t\right\} = \frac{\sum_{(r \in \mathcal{R}, u(f,r) \leq u_{OPT} - t)} \exp\left(\frac{\epsilon u(f,r)}{2\Delta u}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(\frac{\epsilon u(f,r')}{2\Delta u}\right)} \leq \frac{|\mathcal{R}| \exp\left(\frac{\epsilon(u_{OPT} - t)}{2\Delta u}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(\frac{\epsilon u(f,r')}{2\Delta u}\right)} \leq \frac{|\mathcal{R}| \exp\left(\frac{\epsilon(u_{OPT} - t)}{2\Delta u}\right)}{|\mathcal{R}^*| \exp\left(\frac{\epsilon u_{OPT}}{2\Delta u}\right)}$$

$$\leq \frac{|\mathcal{R}|}{|\mathcal{R}^*|} \exp\left(-\frac{\epsilon t}{2\Delta u}\right) \leq |\mathcal{R}| \exp\left(-\frac{\epsilon t}{2\Delta u}\right)$$

Substituting $t = \frac{2\Delta_u}{\varepsilon} \ln\left(\frac{|\mathcal{R}|}{\beta}\right)$ gives us the required equation. □

2.3. Privacy of Streaming Algorithms

2.3.1. Streams and streaming algorithms. Intuitively, a stream is a collection of datasets over time. A natural extension of our notation from datasets to streams is the inclusion of a dimension of time. Thus we represent a stream as a function $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ such that $f(x, t)$ can be interpreted as the number of times the point $x \in \mathcal{X}$ appears at time $t \in \mathbb{N}$. For any $N \subseteq \mathbb{N}$, we will use the notation f_N to denote the restriction of the stream f to the time indices in the set N, that is $f_N : N \times \mathcal{X} \to \mathbb{N}$ such that $f_N(t, x) = f(t, x)$ for all $t \in N$ and $x \in \mathcal{X}$. Similarly, for any time $t \in \mathbb{N}$, $f_t : \mathcal{X} \to \mathbb{N}$ denotes a restriction of f to time t such that $f_t(x) = f(t, x)$ for all $x \in \mathcal{X}$. For example, consider that $\mathcal{X}$ is the set of all possible demographics in a country's voting population. Then we can represent the eligible voters in the country as a stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ where $f(x, t)$ denotes the number of individuals in the population with demographics $x \in \mathcal{X}$ and are eligible to vote at time t.

2.3.1.1. *Differential stream.* Suppose an individual with some demographics $x \in \mathcal{X}$, becomes eligible to vote at some time $t_1 \in \mathbb{N}$. Then, the count $f(x, t_1)$ is incremented by 1 compared to $f(x, t_1 - 1)$ to account for this change. Similarly, if the individual becomes ineligible to vote (for example due to death or giving up on citizenship) at time $t_2 \in \mathbb{N}$ the count $f(x, t_2)$ is decremented by 1 compared to $f(x, t_2 - 1)$. A natural way to keep track of these changes over time if thus with a differential stream. A *differential stream* for f is the stream ∇f defined as,

$$\nabla f(x, t) \coloneqq f(x, t) - f(x, t - 1), \quad t \in \mathbb{N}, \tag{2.8}$$

where we set $f(x, 0) = 0$. The total change of f over all times and points is the quantity

$$\|f\|_\nabla \coloneqq \sum_{x \in \mathcal{X}} \sum_{t \in \mathbb{N}} \left|\nabla f(x, t)\right|, \tag{2.9}$$

which defines a seminorm on the space of data streams.

2.3.1.2. *Streaming algorithm.* As a motivating example, consider an algorithm $\mathcal{A}$ that converts a given input data stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ into a synthetic data stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$. We want this algorithm to be *streaming*: at each time t_0, it can only see the part of the input stream $f(x, t)$ for all $x \in \mathcal{X}$ and $t \le t_0$, and at that time the algorithm outputs $g(x, t_0)$ for all $x \in \mathcal{X}$. Definition 2.3.1 provides a more rigorous definition of a streaming algorithm.

Definition 2.3.1 (Streaming algorithm). *Given an input stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$, an algorithm $\mathcal{A}$ with output stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ is said to be streaming if at any time $t \in \mathbb{N}$ it maps $f_{[t]}$ to g_t, that is, $g(t, x) \coloneqq \mathcal{A}(f_{[t]})(t, x)$ for all $x \in \mathcal{X}$.*

2.3.2. Extending differential privacy to streaming algorithms.

2.3.2.1. *Neighboring streams.* Recall that, to define differential privacy, we need a concept of neighboring inputs that are constructed by omitting the data contributed by a single user. However, a user may have multiple contributions in the stream. Similar to how we handled multiple contributions in a dataset, let us first look at neighboring streams when they differ by a single point.

Definition 2.3.2 (Neighboring streams). *Two streams f and $\tilde{f}$ are said to be neighbors if*

$$\|f - \tilde{f}\|_{\nabla} = 1, \tag{2.10}$$

that is if $\tilde{f}$ can be obtained from f by changing *a single data point*: either one data point is added at some time and is never removed later, or one data point is removed and is never added back later.

2.3.2.2. *Differential privacy for streams.* Using Definition 2.3.2 for neighboring streams we first define the concept of differential privacy for streams that differ in one point.

Definition 2.3.3 (Differential privacy for streams). *A randomized streaming algorithm $\mathcal{A}$ that takes data streams as input is ε-differentially private if for any two neighboring streams f and $\tilde{f}$ that satisfy $\|f - \tilde{f}\| = 1$, the inequality*

$$\mathbb{P}\{\mathcal{A}(\tilde{f}) \in S\} \leq e^{\varepsilon} \cdot \mathbb{P}\{\mathcal{A}(f) \in S\} \tag{2.11}$$

holds for any measurable set of outputs S.

Now that we quantified the effect of the change of a single input data point, we can change any number of input data points. If two data streams satisfy $\|f - \tilde{f}\|_{\nabla} = k$, then applying Definition 2.3.3 k times and using the triangle inequality, we conclude that

$$\mathbb{P}\{\mathcal{A}(\tilde{f}) \in S\} \leq e^{k\varepsilon} \cdot \mathbb{P}\{\mathcal{A}(f) \in S\}.$$

For example, suppose that a patient gets sick and spends a week at some hospital; then she recovers, but after some time she gets sick again and spends another week at another hospital and finally recovers completely. If the data stream $\tilde{f}$ is obtained from f by removing such a patient, then, due to the four events described above, $\|f - \tilde{f}\|_{\nabla} = 4$. Hence, we conclude that

$\mathbb{P}\{\mathcal{A}(\tilde{f}) \in S\} \leq e^{4\varepsilon} \cdot \mathbb{P}\{\mathcal{A}(f) \in S\}$. In other words, the privacy of patients who contribute four events to the data stream is automatically protected as well, although the protection guarantee is four times weaker than for patients who contribute a single event.

2.3.3. Baseline differentially private algorithm for synthetic data generation. In this section, we propose a baseline framework in Algorithm 1 that can extend any offline differentially private algorithm to a streaming differentially private algorithm. Let $\mathcal{A}$ be any differentially private algorithm for generating a synthetic dataset given an input true dataset. The idea is very simple: given an input stream f, at any time t, Algorithm 1 runs an independent instance of algorithm $\mathcal{A}$ on the differential dataset at time t, that is ∇f_t and produces the differential synthetic dataset ∇g_t. We use an instance of this algorithm, with an appropriate subroutine algorithm $\mathcal{A}$, as the baseline in Chapter 3 and 4.

Algorithm 1 A general framework: from differentially private dataset to stream

```
1: Input: An input data stream $f$, a differentially private algorithm $\mathcal{A}$ to generate synthetic
   datasets, the privacy budget $\varepsilon$
2: Output: A synthetic stream $g$.
3: Initialize $g(0, x) = 0$ for all $x \in \mathcal{X}$.
4: for $t = 1, 2, \dots$ do
5:     Set $h_t \leftarrow \mathcal{A}(\nabla f_t, \varepsilon)$
6:     Set $g(t, x) = g(t-1, x) + h_t(x)$ for all $x \in \mathcal{X}$
7:     Release $g_t$.
```

Theorem 2.3.4 (Privacy of Algorithm 1). *If $\mathcal{A}$ satisfies ε-differential privacy, then Algorithm 1 satisfies ε-differential privacy*

Proof of privacy of Algorithm 1. Let $\mathcal{A}^+$ denote the Algorithm 1. For neighboring data streams f and $\tilde{f}$ satisfying Equation 2.10, there exists at most one $\tau \in \mathbb{N}$ and $x \in \mathcal{X}$ such that $\nabla f(x, \tau) \neq \nabla \tilde{f}(x, \tau)$. Since $\mathcal{A}$ is ε-DP, it follows that for any output h_τ,

$$\mathbb{P}\left\{\mathcal{A}(\nabla f_\tau, \varepsilon) = h_\tau\right\} \leq e^{\varepsilon} \cdot \mathbb{P}\left\{\mathcal{A}(\nabla \tilde{f}_\tau, \varepsilon) = h_\tau\right\}.$$

Since at any time $t \neq \tau$, the input streams f and ∇f are identical, it follows that,

$$\mathbb{P}\left\{\mathcal{A}^+(f, \mathcal{A}, \varepsilon) = g\right\} \leq e^{\varepsilon} \cdot \mathbb{P}\left\{\mathcal{A}^+(\tilde{f}, \mathcal{A}, \varepsilon) = g\right\}.$$

Hence, Baseline Algorithm 1 is ε-DP. $\square$

2.4. Counters

A central idea to our streaming algorithms in Chapter 3 and 4 is of *Counters* that are differentially private streaming algorithms for counting values over time. We give a formal definition of a counter in Definition 2.4.1.

Definition 2.4.1. *An (α, δ)-accurate counter $\mathcal{C}$ is a randomized streaming algorithm that estimates the sum of an input stream of values $f : \mathbb{N} \to \mathbb{R}$ and maps it to an output stream of values $g : \mathbb{N} \to \mathbb{R}$ such that for each time $t \in \mathbb{N}$,*

$$\mathbb{P}\left\{\left|g(t) - \sum_{t' \leq t} f(t')\right| \leq \alpha(t, \delta)\right\} \geq 1 - \delta,$$

where the probability is over the randomness of $\mathcal{C}$ and δ is a small constant.

Algorithm 2 Simple counter

```
1: Input: Input stream $f : \mathbb{N} \to \mathbb{R}$, privacy budget $\varepsilon$
2: Output: Output stream $g : \mathbb{N} \to \mathbb{R}$
3: Assume $g(0) \leftarrow 0$
4: Set $g(t) \leftarrow g(t-1) + f(t) + Lap(\frac{1}{\varepsilon})$ for all $t \in \mathbb{N}$
```

Algorithm 3 Block counter with block size B

```
1: Input: Input stream $f : \mathbb{N} \to \mathbb{R}$, privacy budget $\varepsilon$, block size $B$
2: Output: Output stream $g : \mathbb{N} \to \mathbb{R}$
3: $\alpha \leftarrow 0, \beta \leftarrow 0, \beta_{lastblock} \leftarrow 0$
4: for $t = 1, 2, \ldots$ do
5:     $\beta \leftarrow \beta + f(t)$
6:     if $t = k \cdot B$, for some $k \in \mathbb{N}$ then
7:         $\alpha \leftarrow 0,\ \beta \leftarrow \beta + Lap(\frac{2}{\varepsilon})$
8:         $\beta_{lastblock} \leftarrow \beta$
9:     else
10:        $\alpha \leftarrow \alpha + f(t) + Lap(\frac{2}{\varepsilon})$
11:    $g(t) \leftarrow \beta_{lastblock} + \alpha$
```

Furthermore, we are interested in counters that satisfy differential privacy guarantees as per Definition 2.3.3 with respect to the norm $\|f\| := \sum_{t \in \mathbb{N}} |f(t)|$. Note that we have not used the norm $\|\cdot\|_{\nabla}$ here, as the input to the counter algorithm will already be the differential stream ∇f. The foundational work of [12] and [23] introduced private counters for the sum of bit-streams but the same algorithms can be applied to real-valued streams as well. In particular, we will be using the

Algorithm 4 Binary Tree counter

```
1: Input: Input stream $f : \mathbb{N} \to \mathbb{R}$, privacy budget $\varepsilon$, time horizon $T$
2: Output: Output stream $g : \mathbb{N} \to \mathbb{R}$
3: Initialize $\alpha_i, \hat{\alpha}_i \leftarrow 0$ for all $i \in \mathbb{N}$
4: for $t = 1, 2, \ldots$ do
5:     Let $b_n \ldots b_1 b_0$ be the $(n+1)$-bit binary representation of $t$, i.e., $t = \sum_{i=0}^{n} b_i 2^i$
6:     $j \leftarrow \min\{i : b_i \neq 0\}$
7:     $\alpha_j \leftarrow \left(\sum_{i=0}^{j-1} \alpha_i\right) + f(t)$
8:     $\hat{\alpha}_j \leftarrow \alpha_j + Lap(\frac{\log_2 T}{\varepsilon})$
9:     $\alpha_i, \hat{\alpha}_i \leftarrow 0$ for all $i < j$
10:    $g(t) \leftarrow \sum_{i=0}^{n} \hat{\alpha}_i \cdot (\mathbb{1}_{b_i=1})$
```

Simple II, Two-Level, and Binary Tree algorithms from [12], hereafter referred to as Simple, Block, and Binary Tree Counters respectively. We provide the algorithms for Simple, Block, and Binary Tree counter in Algorithms 2, 3, and 4 respectively. As explained in [12], the key principle behind the design of these algorithms is dividing the time horizon into intervals and adding together the *noisy partial sums* from these intervals.

2.4.1. Accuracy of counters. As proved in [12,23] for a fixed failure probability δ, ignoring the constants, the accuracy α at time t for the counters Simple, Block, and Binary Tree is $\mathcal{O}\left(\sqrt{t}\right)$, $\mathcal{O}\left(t^{1/4}\right)$, and $\mathcal{O}\left((\ln t)^{3/2}\right)$ respectively. We provide the rigorous statements in Lemma 2.4.1.1, 2.4.1.2, and 2.4.1.3 respectively. Hence the error in the estimated value in counters grows with time t. The result also suggests that for large time horizons, the Binary Tree algorithm is best to be used as a counter. However, for small values of time t, the bounds suggest that Simple and Block counters are perhaps more effective. We evaluate this hypothesis experimentally and discuss more on which counter is best to be used in Section 3.6.

LEMMA 2.4.1.1. *[Accuracy of Simple Counter] For any input stream $f : \mathbb{N} \to \mathbb{R}$, $t \in \mathbb{N}$, and $\beta > 0$, an ε-differentially private Simple counter is* $\left(\mathcal{O}\left(\frac{1}{\varepsilon}\left(\sqrt{t}\right)\log\frac{1}{\beta}\right), \beta\right)$*-accurate.*

LEMMA 2.4.1.2. *[Accuracy of Block Counter] For any input stream $f : \mathbb{N} \to \mathbb{R}$, $t \in \mathbb{N}$, and $\beta > 0$, an ε-differentially private Simple counter is* $\left(\mathcal{O}\left(\frac{1}{\varepsilon}\left(t^{1/4}\right)\log\frac{1}{\beta}\right), \beta\right)$*-accurate.*

LEMMA 2.4.1.3. *[Accuracy of Binary Tree Counter] For any input stream $f : \mathbb{N} \to \mathbb{R}$, $t \in \mathbb{N}$, and $\beta > 0$, an ε-differentially private Binary tree counter is* $\left(\mathcal{O}\left(\frac{1}{\varepsilon}(\log T)(\sqrt{\log t})\log\frac{1}{\beta}\right), \beta\right)$*-accurate.*

In [12] the authors also provide an algorithm for unbounded streams, termed the Hybird Mechanism, where they run the binary tree mechanism over bounded time intervals of increasing powers of 2. We refer to this algorithm as Unbounded binary tree counter and provide its utility guarantees in Lemma 2.4.1.2.

LEMMA 2.4.1.4. *[Accuracy of unbounded binary tree counter] For any input stream $f : \mathbb{N} \to \mathbb{R}$, $t \in \mathbb{N}$, and $\beta > 0$, an ε-differentially private Unbounded binary tree counter is $\left(\mathcal{O}\left(\frac{1}{\varepsilon}(\log t)^{1.5} \log \frac{1}{\beta}\right), \beta\right)$-accurate.*

2.5. Other variants

Many variants of differential privacy have been considered in the literature. Although we only use the original definition of differential privacy (also called Pure differential privacy), we discuss some other popular variants to highlight how they contrast with our definition.

2.5.1. Local vs central. The first distinction in differential privacy is based on whether or not a central trusted server exists. In Definition 2.1.2 we have assumed the existence of a central server (or curator) that has access to the entire dataset. Indeed it follows from the assumption that neighboring datasets differ by the data of at most one individual. If the central entity curating the dataset cannot be trusted, we must introduce privacy before the data reaches such an entity. Thus, prompting the notion of neighboring datasets at an individual level. The resulting variant is therefore called Local differential privacy (Definition 2.5.1). The definition provided here is inspired by [27]. Note that the input to the algorithm $\mathcal{A}$ in the definition is a point rather than a dataset.

DEFINITION 2.5.1 (Local differential privacy). *A randomized algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{R}$ is said to satisfy ε-local differential privacy if for all pairs $x, \tilde{x} \in \mathcal{X}$ and for any measurable $S \subseteq \mathcal{R}$ we have,*

$$\mathbb{P}\left\{\mathcal{A}(x) \in S\right\} \leq e^{\varepsilon} \cdot \mathbb{P}\left\{\mathcal{A}(\tilde{x}) \in S\right\},$$

where the probability is taken over the randomness of $\mathcal{A}$.

Local differential privacy is a strong privacy guarantee, making it difficult to achieve together with high utility. Indeed the guarantees even hold for any pair of data $x, \tilde{x}$ that may represent extreme information. Geo-indistinguishability [5](Definition 2.5.2) is a generalization of this definition where the bound is based on the distance, say $d(x, \tilde{x})$, between the points x and $\tilde{x}$. Therefore, if the distance $d(x, \tilde{x})$ is large, we may allow for a weaker bound.

Definition 2.5.2 (Geo-indistinguishability). *A randomized algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{R}$ is said to satisfy ε-geo-indistinguishability if for all pairs $x, \tilde{x} \in \mathcal{X}$ and for any measurable $S \subseteq \mathcal{R}$ we have,*

$$\mathbb{P}\left\{\mathcal{A}(x) \in S\right\} \leq e^{\varepsilon \cdot d(x,\tilde{x})} \cdot \mathbb{P}\left\{\mathcal{A}(\tilde{x}) \in S\right\},$$

where the probability is taken over the randomness of $\mathcal{A}$.

Because geo-indistinguishability can be used with any distance metric d, and in particular the Euclidean distance, it has been widely adopted for preserving the privacy of spatial datasets where the curator cannot be trusted. Geo-indistinguishability is also known as *metric privacy.*

2.5.2. Pure vs approximate. Let us redirect our focus to the case when ensuring privacy after a central entity has collected the data. Since differential privacy is a strict notion, most other variants attempt to provide a weaker but meaningful guarantee. (ε, δ)-differential privacy or approx differential privacy (Definition 2.5.3) is among the most popular alternatives. It allows some relaxation when the ε-differential privacy may not hold by having an additive constant. Note that indeed when $\delta = 0$, (ε, δ)-differential privacy is the same as ε-differential privacy.
The differential privacy condition in Definition 2.1.2 can also be seen as,

$$\ln\left(\frac{\mathbb{P}\left\{\mathcal{A}(f) \in S\right\}}{\mathbb{P}\left\{\mathcal{A}(\tilde{f}) \in S\right\}}\right) \leq \varepsilon,$$

for all measurable sets S. Let $Z(S)$ be the random variable equal to the expression $\ln\left(\frac{\mathbb{P}\left\{\mathcal{A}(f) \in S\right\}}{\mathbb{P}\left\{\mathcal{A}(\tilde{f}) \in S\right\}}\right)$ and we refer to it as the *privacy loss.* Then, differential privacy is a worst-case guarantee such that for any S, $Z(S) \leq e$. Other relaxations of differential privacy are based on controlling the tail bounds on Z through its moment-generating function. Rényi differential privacy [59] and zero-concentrated differential privacy [11], Definition 2.5.4 and 2.5.5 respectively, are two such variants.

Definition 2.5.3 ((ε, δ)-differential privacy). *A randomized algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}$ is said to satisfy (ε, δ)-differential privacy (also called approx differential privacy) if for all neighboring datasets*

f and $\tilde{f}$ and for any measurable $S \subseteq \mathcal{R}$ we have,

$$\mathbb{P}\left\{\mathcal{A}(f) \in S\right\} \le e^{\varepsilon} \cdot \mathbb{P}\left\{\mathcal{A}(\tilde{f}) \in S\right\} + \delta,$$

where the probability is taken over the randomness of $\mathcal{A}$.

Definition 2.5.4 (Rényi differential privacy). *Given $\alpha > 1$, a randomized algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}$ is said to satisfy (α, ε)-rényi differential privacy if for all neighboring datasets f and $\tilde{f}$,*

$$D_\alpha\left(\mathcal{A}(f) \middle\| \mathcal{A}(\tilde{f})\right) \le \varepsilon,$$

where D_α is the Rényi divergence of order α and the probability is taken over the randomness of $\mathcal{A}$.

Definition 2.5.5 (Concentrated differential privacy). *A randomized algorithm $\mathcal{A} : \mathbb{N}^{\mathcal{X}} \to \mathcal{R}$ is said to satisfy (ξ, ρ)-zero-concentrated differential privacy (also called approx differential privacy) if for all neighboring datasets f and $\tilde{f}$ and for any $\alpha \in (1, \infty)$ we have,*

$$D_\alpha\left(\mathcal{A}(f) \middle\| \mathcal{A}(\tilde{f})\right) \le \xi + \rho\alpha,$$

where D_α is the Rényi divergence of order α and the probability is taken over the randomness of $\mathcal{A}$.

We refer the reader to [**17**] for a detailed comparison and relation between various variants of the definition of differential privacy.

CHAPTER 3

Differentially Private Synthetic Spatial Stream

3.1. Motivation and problem setup

In this chapter, we present an algorithm that transforms a data stream into a differentially private data stream. The algorithm can handle quite general data streams: at each time $t \in \mathbb{N}$, the data is a subset of some abstract set $\mathcal{X}$. For instance, if $\mathcal{X}$ can be the location of all U.S. hospitals, and the data at time $t = 3$ can be the locations of all patients spending time in hospitals on day 3. Such data can be conveniently represented by a *data stream*, which is any function of the form $f(x, t) : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$. For instance, $f(x, 3)$ can be the number of COVID positive patients at location x on day 3.

We present *an ε-differentially private, streaming algorithm that takes as an input a data stream and returns as an output a data stream.* This algorithm tries to make the output stream as close as possible to the input data stream, while upholding differential privacy.

3.2. A macroscopic overview of our method: PHDStream

Here, we describe our method in broad brushstrokes. In Section 3.4 we discuss how we optimize the computational and storage cost of our algorithm.

3.2.1. From streaming on sets to streaming on trees. First, we convert the problem of differentially private streaming of a function $f(x, t)$ on a set $\mathcal{X}$ to a problem of differentially private streaming of a function $F(x, t)$ on a *tree*. To this end, fix some *hierarchical partition* of the domain $\mathcal{X}$. Thus, assume that $\mathcal{X}$ is partitioned into some $\beta > 1$ subsets $\mathcal{X}_1, \ldots, \mathcal{X}_\beta$, and each of these subsets $\mathcal{X}_i$ is partitioned into β further subsets, and so on. A hierarchical partition can be equivalently represented by a *tree* T whose vertices are subsets of $\mathcal{X}$ and the children of each vertex form a partition of that vertex. Thus, the tree T has root $\mathcal{X}$; the root is connected to the β vertices $\mathcal{X}_1, \ldots, \mathcal{X}_\beta$, and so on. We refer to β as the *fanout* number of the tree.

In practice, there often exists a natural hierarchical decomposition of $\mathcal{X}$. For example, if $\mathcal{X} = \{0,1\}^d$ a binary partition obtained by fixing a coordinate is natural such as $\mathcal{X}_1 = \{0\} \times \{0,1\}^{d-1}$ and $\mathcal{X}_2 = \{1\} \times \{0,1\}^{d-1}$. Each of $\mathcal{X}_1$ and $\mathcal{X}_2$ can be further partitioned by fixing another coordinate. As another example, if $\mathcal{X} = [0,1]^d$ and fanout $\beta = 2$, a similar natural partition can be obtained by splitting a particular dimension's region into halves such that $\mathcal{X}_1 = [0, \frac{1}{2}) \times [0,1]^{d-1}$ and $\mathcal{X}_2 = [\frac{1}{2}, 1] \times [0,1]^{d-1}$. Each of $\mathcal{X}_1$ and $\mathcal{X}_2$ can be further partitioned by splitting another coordinate's region into halves.

We can convert any function on the set $\mathcal{X}$ into a function on the vertices of the tree T by summing the values in each vertex. I.e., to convert $f \in \mathbb{R}^{\mathcal{X}}$ into $F \in \mathbb{R}^{V(T)}$, we set

$$F(v) \coloneqq \sum_{x \in v} f(x), \quad v \in V(T). \tag{3.1}$$

Vice versa, we can convert any function G on the vertices of the tree T into a function g on the set $\mathcal{X}$ by assigning value $G(v)$ to one arbitrarily chosen point in each leaf v. In practice, however, the following variant of this rule works better if $G(v) > 0$. Assign value 1 to $\lceil G(v) \rceil$ random points in v, i.e. set

$$g \coloneqq \sum_{v \in \mathcal{L}(T)} \sum_{i=1}^{\lceil G(v) \rceil} \mathbb{1}_{x_i(v)} \tag{3.2}$$

where $x_i(v)$ are independent random points in v and $\mathbb{1}_x$ denotes the indicator function of the set $\{x\}$. Here, $\mathcal{L}(T)$ is a collection of the leaf nodes in T. The points $x_i(v)$ can be sampled from any probability measure on v, and in practice we often choose the uniform measure on v.

Summarizing, we reduced our original problem to constructing an algorithm that transforms any given stream $F(x,t) : V(T) \times \mathbb{N} \to \{0,1,2,\ldots\}$ into a differentially private stream $G(x,t) : V(T) \times \mathbb{N} \to \{0,1,2,\ldots\}$ where $V(T)$ is the vertex set of a fixed, known tree T.

3.2.2. Consistent extension. Let $C(T)$ denote the set of all functions $F \in \mathbb{R}^{V(T)}$ that can be obtained from functions $f \in \mathbb{R}^{\mathcal{X}}$ using transformation (3.1). The transformation is linear, so $C(T)$ must be a linear subspace of $\mathbb{R}^{V(T)}$. A moment's thought reveals that $C(T)$ is comprised of all *consistent functions* – the functions F that satisfy the equations

$$F(v) = \sum_{u \in \text{children}(v)} F(u) \quad \text{for all } v \in V(T). \tag{3.3}$$

Any function on $V(T)$ can be transformed into a consistent function by pushing the values up the tree and spreading them uniformly down the tree. More specifically, this can be achieved by the linear transformation

$$\mathrm{Ext}_T : \mathbb{R}^{V(T)} \to C(T),$$

that we call the *consistent extension*. Suppose that a function F takes value 1 on some vertex $v \in V(T)$ and value 0 on all other vertices. To define $G = \mathrm{Ext}_T(F)$, we let $G(u)$ equal 1 for any ancestor of v including v itself, $1/\beta$ for any child of v, $1/\beta^2$ for any grandchild of v, and so on. In other words, we set $G(u) = \beta^{-\max(0,d(v,u))}$ where $d(v,u)$ denotes the directed distance on the tree T, which equals the usual graph distance (the number of edges in the path from v to u) if u is a descendant of v, and minus the graph distance otherwise. Extending this rule by linearity, we arrive at the explicit definition of the consistent extension operator:

$$\mathrm{Ext}_T(F)(u) \coloneqq \sum_{v \in V(T)} F(v)\, \beta^{-\max(0,d(v,u))}, \quad u \in V(T).$$

By definition (3.3), a consistent function is uniquely determined by its values on the leaves of the tree T. Thus a natural norm of $C(T)$ is

$$\|F\|_{C(T)} \coloneqq \sum_{v \in \mathcal{L}(T)} \left|F(v)\right|. \tag{3.4}$$

3.2.3. Differentially private tree. A key subroutine of our method is a version of the remarkable algorithm PrivTree due to [79]. We present this version in Algorithm 5. In the absence of noise addition, one can think of PrivTree as a deterministic algorithm that inputs a tree T and a function $F \in \mathbb{R}^{V(T)}$ and outputs a subtree of T. The algorithm grows the subtree iteratively: for every vertex v, if $F(v)$ is larger than a certain threshold θ, the children of vertex v are added to the subtree.

3.2.4. Differentially private stream. We present our method PHDStream (Private Hierarchical Decomposition of Stream) in Algorithm 7. Algorithm 6 transforms an input stream $F(\cdot,t) \in V(T) \times \mathbb{N} \to \mathbb{R}$ into a stream $G(\cdot,t) \in V(T) \times \mathbb{N} \to \mathbb{R}$. In the algorithm, $\mathcal{L}(T)$ denotes the set of leaves of tree T, and $\mathrm{Lap}(\cdot, t, 2/\varepsilon)$ denote independent Laplace random variables. Using Algorithm 6 as a subroutine, Algorithm 7 transforms a stream $f(x,t) \in \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ into a stream $g(x,t) \in \mathcal{X} \times \mathbb{N} \to \mathbb{R}$:

Algorithm 5 $\mathrm{PrivTree}_T$

```
1: Input: $F \in C(T)$, the privacy budget parameter $\varepsilon$, the threshold count for a node $\theta$.
2: Output: A subset $S$ of the tree $T$.
3: Set $\lambda \leftarrow \frac{2\beta-1}{\beta-1} \cdot \frac{2}{\varepsilon}$ and $\delta \leftarrow \lambda \ln \beta$.
4: Initialize an empty tree $S$.
5: $U \leftarrow \{root(T)\}$; denotes a set of un-visited nodes.
6: for $v \in U$ do
7:     Add $v$ to $S$.
8:     $U \leftarrow U \setminus \{v\}$; mark $v$ visited.
9:     $b(v) \leftarrow F(v) - depth(v) \cdot \delta$; is a biased aggregate count of the node $v$.
10:    $b(v) \leftarrow \max \{b(v), \theta - \delta\}$; to ensure that $b(v)$ is not too small.
11:    $\tilde{b}(v) \leftarrow b(v) + Lap(\lambda)$; noisy version of the biased count.
12:    if $(\tilde{b}(v) > \theta)$; noisy biased count is more than threshold then
13:        Add children of $v$ in $T$ to $U$.
```

Algorithm 6 PHDStreamTree

```
1: Input: $F \in C(T)$, the privacy budget parameter $\varepsilon$, the threshold count for a node $\theta$.
2: Output: A stream $G \in C(T)$.
3: Initialize $G(v, 0) = 0$ for all $v \in V(T)$.
4: for every time $t \in \mathbb{N}$ do
5:     $T(t) \leftarrow \mathrm{PrivTree}_T\big(G(\cdot, t-1) + \nabla F(\cdot, t), \varepsilon/2, \theta\big)$.
6:     $d(v,t) \leftarrow \begin{cases} \nabla F(v,t) + \mathrm{Lap}(\cdot, t, 2/\varepsilon) & \text{if } v \in \mathcal{L}(T(t)) \\ 0 & \text{otherwise.} \end{cases}$
7:     $G(v,t) \leftarrow G(v, t-1) + \mathrm{Ext}_T(d(\cdot, t)) \quad \forall v \in V(T)$.
```

Algorithm 7 PHDStream

```
1: Input: Input data stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$, the privacy budget parameter $\varepsilon$, the threshold count
   for a node $\theta$.
2: Output: Data stream $g$.
3: Apply PHDStreamTree (Algorithm 6) for $F$ obtained from $f$ using (3.1), and we convert the
   output $G$ into $g$ using (3.2).
```

In the first step of this algorithm, we consider the previously released synthetic stream $G(\cdot, t-1)$, update it with the newly received real data $\nabla F(\cdot, t)$, and feed it into $\mathrm{PrivTree}_T$, which produces a differentially private subtree $T(t)$ of T.

In the next two steps, we compute the updated stream on the tree. It is tempting to choose for $G(\cdot, t)$ a stream computed by a simple random perturbation as, $G(\cdot, t) = G(\cdot, t-1) + \nabla F(\cdot, t) + \mathrm{Lap}(\cdot, t, 2/\varepsilon)$. The problem, however, is that such randomly perturbed stream would not be differentially private. Indeed, imagine we make a stream $\tilde{f}$ by changing the value of the input stream f at some time $t \in \mathbb{N}$ and point $x \in \mathcal{X}$ by 1. Then the sensitivity condition in Definition 2.3.2 holds. But when we convert f into a consistent function F on the tree, using (3.1), that little change propagates up the tree. It affects the values of F not just at one leaf $v \ni x$ but all of the ancestors of v as well, and this could be too many changes to protect. In other words, consistency on the tree makes sensitivity too high, which in turn jeopardizes privacy.

To halt the propagation of small changes up the tree, the last two steps of the algorithm restrict the function *only* on the leaves of the subtree. This restriction controls sensitivity: a change to $F(v)$ made in one leaf v does not propagate anymore, and the resulting function $d(v,t)$ is differentially private. In the last step, we extend the function $d(v,t)$ from the leaves to the whole tree–an operation that preserves privacy–and use it as an update to the previous, already differentially private, synthetic stream $G(\cdot, t-1)$.

These considerations lead to the following privacy guarantee as announced in Section 2.3.2, that is in the sense of Definition 2.3.3, for the $\|\cdot\|_\nabla$ norm.

Theorem 3.2.1 (Privacy of PHDStream). *The PHDStream algorithm is ε-differentially private.*

We provide a rigorous analysis of privacy in the next Section 3.3.

3.3. Privacy of PHDStream

In this section, we will first discuss the privacy of $\mathrm{PrivTree}_T$ (Algorithm 5). Then using it together with the privacy of PHDStreamTree (Algorithm 6) we provide the privacy of our overall method PHDStream (Algorithm 7).

3.3.1. Privacy of $\mathrm{PrivTree}_T$.

Theorem 3.3.1 (Privacy of *$\mathrm{PrivTree}_T$*). *The randomized algorithm $\mathcal{M} := \mathrm{PrivTree}_T(F, \varepsilon, \theta)$ is ε-differentially private in the $\|\cdot\|_{C(T)}$ norm for any $\theta \geq 0$.*

Proof of Theorem 3.3.1. Similar to [79] we define the following two functions:

$$\rho_{\lambda,\theta}(x) = \ln\left(\frac{\mathbb{P}\left\{x + Lap(\lambda) > \theta\right\}}{\mathbb{P}\left\{x - 1 + Lap(\lambda) > \theta\right\}}\right), \tag{3.5}$$

$$\rho^{\top}_{\lambda,\theta}(x) = \begin{cases} 1/\lambda, & x < \theta + 1, \\ \frac{1}{\lambda}\exp\left(\frac{\theta+1-x}{\lambda}\right), & \textit{otherwise}. \end{cases} \tag{3.6}$$

It can be shown that $\rho_{\lambda,\theta}(x) \leq \rho^{\top}_{\lambda,\theta}(x)$ for any x.

Let us consider the neighbouring functions F and $\tilde{F}$ such that $\left\|F - \tilde{F}\right\|_{C(T)} = 1$ and there exists $v_* \in V(T)$ such that $F(v_*) \neq \tilde{F}(v_*)$. Consider the output S, a subtree of T.

Case 1: $F(v_*) > \tilde{F}(v_*)$

Note that there will be exactly one leaf node in S that differs in the count for the functions F and $\tilde{F}$. Let $v_1, v_2, \ldots, v_k$ be the path from the root to this leaf node. As per Algorithm 5, let us denote the calculated biased counts of any node $v \in S$ for the functions F and $\tilde{F}$ as $b(v)$ and $\tilde{b}(v)$, respectively. Then, for any $v \in S$, we have the following relations,

$$\tilde{F}(v) = \begin{cases} F(v) - 1, & v \in \{v_1, v_2, v_3, \ldots, v_k\} \\ F(v), & \textit{otherwise}, \end{cases}$$

$$\tilde{b}(v) = \begin{cases} b(v) - 1, & v \in \{v_1, v_2, v_3, \ldots, v_k\}, b(v) > \theta - \delta, \\ b(v), & \textit{otherwise}. \end{cases}$$

Let there exist an $m \in [k-1]$, s.t. $b(v_m) \geq \theta - \delta + 1$ and $b(v_{m+1}) = \theta - \delta$. We then show that there is a difference in count by at least δ between parent and child for all $i \in [2, m]$

$$b(v_i) = F(v_i) - \delta \cdot \textit{depth}(v) \leq F(v_{i-1}) - \delta \cdot (\textit{depth}(v_{i-1}) + 1) \leq b(v_{i-1}) - \delta.$$

Thus we have,

$$\begin{cases} b(v_{i-1}) \geq b(v_i) + \delta \geq \theta + 1, & i \in [2, m], \\ b(v_i) = \theta - \delta, & \textit{otherwise}. \end{cases} \tag{3.7}$$

Finally, we can show DP as follows,

$$\begin{aligned}
\ln\left(\frac{\mathbb{P}\left\{\mathcal{M}(F)=S\right\}}{\mathbb{P}\left\{\mathcal{M}(\tilde{F})=S\right\}}\right) &= \sum_{i=1}^{k-1}\ln\left(\frac{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_i)+Lap(\lambda)>\theta\right\}}\right)+\ln\left(\frac{\mathbb{P}\left\{b(v_k)+Lap(\lambda)\le\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_k)+Lap(\lambda)\le\theta\right\}}\right)\\
&= \sum_{i=2}^{m}\ln\left(\frac{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}{\mathbb{P}\left\{b(v_i)-1+Lap(\lambda)>\theta\right\}}\right)+\sum_{i=m+1}^{k-1}\ln\left(\frac{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}\right)\\
&\quad+\ln\left(\frac{\mathbb{P}\left\{b(v_k)+Lap(\lambda)\le\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_k)+Lap(\lambda)\le\theta\right\}}\right)\\
&\quad\text{[Using the fact that } \ln x\le 0 \text{ for all } x\le 1]\\
&\le \sum_{i=2}^{m}\ln\left(\frac{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}{\mathbb{P}\left\{b(v_i)-1+Lap(\lambda)>\theta\right\}}\right)+0+0.
\end{aligned}$$

Using equations (3.5) and (3.6) we have,

$$\begin{aligned}
\ln\left(\frac{\mathbb{P}\left\{\mathcal{M}(F)=S\right\}}{\mathbb{P}\left\{\mathcal{M}(\tilde{F})=S\right\}}\right) &\le \sum_{i=2}^{m}\rho_{\lambda,\theta}(b(v_i))\le\sum_{i=2}^{m}\rho^{\top}_{\lambda,\theta}(b(v_i))=\rho^{\top}_{\lambda,\theta}(b(v_m))+\sum_{i=2}^{m-1}\rho^{\top}_{\lambda,\theta}(b(v_i))\\
&=\frac{1}{\lambda}+\sum_{i=2}^{m-1}\frac{1}{\lambda}\exp\left(\frac{\theta+1-b(v_i)}{\lambda}\right)=\frac{1}{\lambda}+\sum_{i=1}^{m-2}\frac{1}{\lambda}\exp\left(\frac{\theta+1-b(v_{m-i})}{\lambda}\right)\\
&\quad\text{[Using equation (3.7) and } b(v_{m-1})\ge b(v_m)+\delta\ge\theta+1]\\
&\le\frac{1}{\lambda}+\frac{1}{\lambda}\sum_{i=1}^{m-2}e^{-\delta/\lambda}\le\frac{1}{\lambda}+\frac{1}{\lambda}\cdot\frac{1}{1-e^{-\delta/\lambda}}=\frac{1}{\lambda}\cdot\frac{2e^{\delta/\lambda}-1}{e^{\delta/\lambda}-1}=\varepsilon.
\end{aligned}$$

Case 2: $F(v_*)<\tilde{F}(v_*)$

Following the same notations as Case 1, for any $v\in S$, we have the following relations,

$$\tilde{F}(v)=\begin{cases}F(v)+1, & v\in\left\{v_1,v_2,v_3,\dots,v_k\right\},\\ F(v), & \textit{otherwise},\end{cases}$$

$$\tilde{b}(v)=\begin{cases}b(v)+1, & v\in\left\{v_1,v_2,v_3,\dots,v_k\right\}, b(v)\ge\theta-\delta,\\ b(v), & \textit{otherwise}.\end{cases}$$

Finally, we can show DP as follows,

$$\ln\left(\frac{\mathbb{P}\left\{\mathcal{M}(F)=S\right\}}{\mathbb{P}\left\{\mathcal{M}(\tilde{F})=S\right\}}\right)=\sum_{i=1}^{k-1}\ln\left(\frac{\mathbb{P}\left\{b(v_i)+Lap(\lambda)>\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_i)+Lap(\lambda)>\theta\right\}}\right)+\ln\left(\frac{\mathbb{P}\left\{b(v_k)+Lap(\lambda)\leq\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_k)+Lap(\lambda)\leq\theta\right\}}\right)$$

$$[\text{Since } \ln x \leq 0 \text{ for all } x \leq 1]$$

$$\leq 0+\ln\left(\frac{\mathbb{P}\left\{b(v_k)+Lap(\lambda)\leq\theta\right\}}{\mathbb{P}\left\{\tilde{b}(v_k)+1+Lap(\lambda)\leq\theta\right\}}\right)\leq\varepsilon.$$

Hence the algorithm is ε-differentially private.

□

Proof of Theorem 3.2.1. It suffices to show that PHDStreamTree is ε-differentially private, since PHDStream is doing a post-processing on its output which is independent of the input data stream. Let f and $\tilde{f}$ be neighboring data streams such that $\left\|f-\tilde{f}\right\|_{\nabla}=1$. Let F and $\tilde{F}$ be the corresponding streams on the vertices of T respectively. Since f and $\tilde{f}$ are neighbors, there exists $\tau\in\mathbb{N}$ such that $\left\|\nabla F(\cdot,\tau)-\nabla\tilde{F}(\cdot,\tau)\right\|_{C(T)}=1$. Note that, at any time t, PHDStreamTree only depends on ∇F_t from the input data stream. Hence for the differential privacy over the entire stream, it is sufficient to show that the processing at time τ is differentially private. Note that both PrivTree$_T$ and the calculation of $d(\cdot,t)$ satisfy $\varepsilon/2$-differential privacy by Theorem 3.3.1 and by the Laplace Mechanism, respectively. Hence their combination, that is PHDStreamTree at time τ, satisfies ε-differential privacy. At any $t\neq\tau$, $\nabla F(v,t)=\nabla\tilde{F}(v,t)$ for all $v\in T$. Hence PHDStreamTree satisfies ε-differential privacy for the entire input stream F. □

3.4. Optimizing the algorithm

In Algorithm 6, at any time t, we have to perform computations for every node $v\in T$. Since T can be a tree with large depth and the number of nodes is exponential with respect to depth, the memory and time complexity of PHDStreamTree, as described in Algorithm 6 is also exponential in depth of the tree T. Note however that we do not need the complete tree T, and can limit all computations to the subtree $T(t)$ as selected by PrivTree at time t. Based on this idea, we propose a version of PHDStream in Algorithm 8 which is storage and runtime efficient.

Algorithm 8 Compute efficient PHDStream

```
1: Input: Input data stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$, a fixed and known tree $T$, the privacy budget parameter $\varepsilon$, the threshold count for a node $\theta$.
2: Output: A synthetic data stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$.
3: Set $\lambda \leftarrow \frac{2\beta-1}{\beta-1} \cdot \frac{2}{\varepsilon}$ and $\delta \leftarrow \lambda \ln \beta$.
4: Initialize $C_a$, $C_n$, and $C_d$ and make them accessible to Algorithm 9.
5: for every time $t \in \mathbb{N}$ do
6:     Set $T(t)$ as the output of Algorithm 9 with parameters $(\nabla f(\cdot, t), \delta, \theta, \lambda, \varepsilon/2)$.
7:     Calculate $G(v) \leftarrow C_a(v) + C_n(v) + C_d(v)$ for all $v \in leaves(T(t))$.
8:     Convert $G$ to $g(\cdot, t)$ as per equation 3.2.
```

Algorithm 9 Modified PrivTree$_T$ subroutine

```
1: Input: Count of points in $\mathcal{X}$ as $h : \mathcal{X} \to \mathbb{R}$, parameters related to privacy $\delta, \theta, \lambda, \varepsilon'$.
2: Output: A subtree $T^*$ of $T$.
3: Initialize an empty subtree $T^*$ of $T$.
4: Initialize a queue $Q$ of un-visited nodes.
5: Push $root(T)$ to $Q$.
6: Initialize an empty stack $S$.
7: while $Q$ is not empty do
8:     Pop $v$ from $Q$.
9:     Add $v$ to the subtree $T^*$.
10:    Find $H(v)$ using $h$ as per equation 3.1.
11:    To enforce ancestor consistency, update
12: $C_a(v) \leftarrow \frac{1}{\beta} \cdot \big(C_a(parent(v)) + C_n(parent(v))\big)$.
13:    Set $s(v) \leftarrow C_a(v) + C_n(v) + C_d(v)$
14:    Set $b(v) \leftarrow \big(s(v) + H(v) - depth(v) \cdot \delta\big)$; as biased count of the node $v$.
15:    $b(v) \leftarrow \max\big\{b(v), \theta - \delta\big\}$; to ensure that $b(v)$ is not too small.
16:    $\hat{b}(v) \leftarrow b(v) + Lap(\lambda)$; noisy version of the biased count.
17:    if $\Big(\hat{b}(v) > \theta\Big)$ and $children(v)$ is not empty in $T$ then
18:        Push $w$ to $Q$ for all $w \in children(v)$ in $T$.
19:        Push $v$ to $S$.
20:    else
21:        Update $C_n(v) \leftarrow C_n(v) + H(v) + Lap(2/\varepsilon')$ .
22: while $S$ is not empty do
23:    Pop $v$ from $S$.
24:    To enforce descendant consistency, update
25: $C_d(v) \leftarrow \sum_{w \in children(v)} \big(C_d(w) + C_n(w)\big)$.
```

Note that in Algorithm 6, at any time $t \in \mathbb{N}$, we find the differential synthetic count of any node v as $d(v,t)$ only if v is a leaf in $T(t)$. To maintain the consistency, as given by equation 3.3, this count gets pushed up to ancestors or spreads down to the descendants by the consistency extension operator Ext_T. A key challenge in efficiently enforcing consistency is that we do not want to process nodes that are not present in $T(t)$. Since the operator Ext_T is linear if we have the total differential synthetic count of all nodes v till time t, that is $\sum_{t' \leq t} d(v, t')$, we can find the count of any node resulting after consistency extension. We track this count in a function $C_n : V(T) \to \mathbb{R}$ such that at any time t, $C_n(v)$ represents the total differential synthetic count of node v for times $t' \leq t$ when $v \in \mathit{leaves}(T(t'))$. We update the value $C_n(v)$ for any node $v \in \mathit{leaves}(T(t))$ as,

$$C_n(v) \leftarrow C_n(v) + \left(\nabla F(v) + \mathit{Lap}(2/\varepsilon)\right).$$

Furthermore, to efficiently enforce consistency, we introduce C_a and C_d mappings $V(T) \to \mathbb{R}$. At any time t, $C_a(v)$ and $C_d(v)$ denote the count a node v has received at time t when enforcing consistency from its ancestors and descendants, respectively. Consistency due to counts being passed down from ancestors can be enforced with the equation

$$C_a(v) = \frac{1}{\beta} \cdot \left(C_a(\mathit{parent}(v)) + C_n(\mathit{parent}(v))\right). \tag{3.8}$$

Consistency due to counts being pushed up from descendants can be enforced with the equation

$$C_d(v) = \sum_{w \in \mathit{children}(v)} \left(C_d(w) + C_n(w)\right). \tag{3.9}$$

We assume that $C_a(v) = C_n(v) = C_d(v) = 0$ for all $v \in V(T)$ at the beginning of Algorithm 8. With the help of these functions, the synthetic count of any node $v \in T(t)$ can be calculated as $s(v) = C_a(v) + C_n(v) + C_d(v)$.

Algorithm 9 is a modified version of PrivTree Algorithm 5. We use it as a subroutine of Algorithm 8, where at time t it is responsible for creating the subtree $T(t)$ and updating the values of the functions C_a, C_n, and C_d.

The algorithm uses two standard data structures called *Stack* and *Queue* which are ordered sets where the order is based on the time at which the elements are inserted. We interact with the data structures using two operations *Push* and *Pop*. Push is used to insert an element, whereas pop is used to remove an element. The key difference between a Queue and a Stack is that the operation pop on a Queue returns the element inserted earliest but on a Stack returns the element inserted latest. Thus Queue and Stack follow the FIFO (first-in, first-out) and LIFO (last-in, first-out) strategy, respectively.

3.4.1. Privacy of compute-efficient PHDStream. At any time $t \in \mathbb{N}$, the key difference of Algorithm 8 (compute efficient PHDStream) from Algorithm 7 (PHDStream) is that (1) it does not compute $\nabla F(v,t)$ for any node $v \in T \setminus T(t)$, and (2) it does not use the consistency operator Ext_T and instead relies on the mappings C_a, C_n, and C_d. Note that none of these changes affect the constructed tree $T(t)$ and the output stream $g(\cdot,t)$. Since both algorithms are equivalent in their output and Algorithm 7 is ε-differentially private, we conclude that Algorithm 8 is also ε-differentially private.

3.5. Counters and selective counting

A key step of Algorithm 6 at any time t is Step 6 where we add noise to the leaves of the subtree $T(t)$. Consider a node $v \in V(T)$ that becomes a leaf in the subtree $T(t)$ for times $t \in N_v \subseteq N$. In the algorithm, we add an independent noise at each time in N_v. Focusing only on a particular node, can we make this counting more efficient? The question becomes: given an input stream of values for a node v as $\nabla F(v,t)$ for $t \in N_v$, can we find an output stream of values $d(v,\cdot)$ in a differentially private manner while ensuring that the input and output streams are close to each other? This problem can be solved using Counters as discussed in Section 2.4.

3.5.1. PHDStreamTree with counters. Algorithm 10 is a version of Algorithm 6 with counters. Here, we create an instance of some counter algorithm for each node $v \in V(T)$. Algorithm 10 is agnostic to the counter algorithm used, and we can even use different counter algorithms for different nodes of the tree. Since at any time $t \in \mathbb{N}$, we only count at the leaf nodes of the tree $T(t)$, the counter C_v is updated for any node $v \in V(T)$ if and only if $v \in \mathcal{L}(T(t))$. Hence the input to the counter C_v is the restriction of the stream ∇F on the set of times in N_v, that is $\nabla F(v,\cdot)\big|_{N_v}$, where $N_v = \{t \in \mathbb{N} \mid v \in \mathcal{L}(T(t))\}$. In the subsequent sections, we discuss a few general ideas about the use of multiple counters and selectively updating some of them. We use these discussions to prove the privacy of Algorithm 10 in Subsection 3.5.5.

3.5.2. Multi-dimensional counter. To efficiently prove the privacy guarantees of our algorithm, which utilizes many counters, we introduce the notion of a multi-dimensional counter. A d-dimensional counter $\mathcal{C}$ is a randomized streaming algorithm consisting of d independent counters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_d$ such that it maps an input stream $f : \mathbb{N} \to \mathbb{R}^d$ to an output stream $g : \mathbb{N} \to \mathbb{R}^d$ with $g(t) = \big(\mathcal{C}_1(f(t)_1), \mathcal{C}_2(f(t)_2), \ldots, \mathcal{C}_d(f(t)_d)\big)$ for all $t \in \mathbb{N}$. For differential privacy of such counters, we

Algorithm 10 PHDStreamTree with counters

```
1: Input: $F \in C(T)$, privacy budget parameter $\varepsilon$, threshold count for a node $\theta$.
2: Output: A stream $G \in C(T)$.
3: Initialize $G(v, 0) = 0$ for all $v \in V(T)$.
4: Initialize a counter $\mathcal{C}_v$ with privacy budget $\varepsilon/2$ for all nodes $v \in V(T)$.
5: for every time $t \in \mathbb{N}$ do
6:     $T(t) \leftarrow \text{PrivTree}_T\big(G(\cdot, t-1) + \nabla F(\cdot, t), \varepsilon/2, \theta\big)$.
7:     $d(v,t) \leftarrow \begin{cases} \mathcal{C}_v(\nabla F(v,t)) & \text{if } v \in \mathcal{L}(T(t)) \\ 0 & \text{otherwise.} \end{cases}$
8:     $G(v,t) \leftarrow G(v, t-1) + \text{Ext}_T(d(\cdot, t)) \quad \forall v \in V(T)$.
```

will still use Definition 2.3.3 but with the extension of the norm to streams with multi-dimensional output as $\|f\| := \sum_{t \in \mathbb{N}} \big\|f(t)\big\|_{\ell_1}$.

3.5.3. Selective Counting. Let us assume that we have a set of k counters $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$. At any time $t \in \mathbb{N}$, we want to activate exactly one of these counters as selected by a randomized streaming algorithm $\mathcal{M}$. The algorithm $\mathcal{M}$ depends on the input stream f and optionally on the entire previous output history, that is the time indices when each of the counters was selected and their corresponding outputs at those times. We present this idea formally as Algorithm 11.

Algorithm 11 Online Selective Counting

```
1: Input: An input stream $f$, a set $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of $k$ counters, a counter-selecting differentially
   private algorithm $\mathcal{M}$.
2: Output: An output stream $g$.
3: Initialize $J_l \leftarrow \emptyset, \forall\, l \in \{1, 2, \ldots, k\}$.
4: for $t = 1$ to $\infty$ do
5:     $l_t \leftarrow \mathcal{M}(f(t), g(t-1))$.
6:     $J_{l_t} \leftarrow J_{l_t} \cup \{t\}$; Add time index $t$ for counter $l_t$.
7:     Set $g(t) \leftarrow \big(l_t, \mathcal{C}_{l_t}(f(t))\big)$; Selected counter and updated count.
```

3.5.4. Privacy of selective counting.

THEOREM 3.5.1. *(Selective Counting) Let $\mathcal{C}_i, i \in \{1, 2, \ldots, k\}$ be ε_C-DP. Let $\mathcal{M}$ be a counter-selecting streaming algorithm that is ε_M-DP. Then, Algorithm 11 is $(\varepsilon_M + \varepsilon_C)$-DP.*

PROOF. Let us denote Algorithm 11 as $\mathcal{A}$. Let f and $\tilde{f}$ be neighboring data streams with $\left\|f - \tilde{f}\right\| = 1$. Without loss of generality, there exists $\tau \in \mathbb{N}$ such that $f(\tau) = \tilde{f}(\tau) + 1$. Let g be the output stream we are interested in. Let $f|_{[t]}$ denote a restriction of the stream f until time t for any $t \in \mathbb{N}$.

Since the input streams are identical till time τ, we have,

$$\mathbb{P}\left\{\mathcal{A}\left(f|_{[\tau-1]}\right) = g|_{[\tau-1]}\right\} = \mathbb{P}\left\{\mathcal{A}\left(\tilde{f}\Big|_{[\tau-1]}\right) = g|_{[\tau-1]}\right\}$$

At time τ, let $g(\tau) = (l_\tau, m_\tau)$. Since $\mathcal{M}$ is ε_M-DP, we have,

$$\mathbb{P}\left\{\mathcal{M}\left(f|_{[\tau]}, g|_{[\tau-1]}\right) = l_\tau\right\} \le e^{\varepsilon_M} \cdot \mathbb{P}\left\{\mathcal{M}\left(\tilde{f}\Big|_{[\tau]}, g|_{[\tau-1]}\right) = l_\tau\right\}.$$

Moreover, since $\mathcal{M}_{l_\tau}$ is ε_C-DP,

$$\mathbb{P}\left\{\mathcal{C}_{l_\tau}\left(f|_{J_{l_t}}\right) = m_\tau\right\} \le e^{\varepsilon_C} \cdot \mathbb{P}\left\{\mathcal{C}_{l_\tau}\left(\tilde{f}\Big|_{J_{l_t}}\right) = m_\tau\right\}.$$

Combining the above two we have,

$$\begin{aligned}
&\mathbb{P}\left\{\mathcal{A}\left(f|_{[\tau]}\right) = g(\tau)\right\} \\
&= \mathbb{P}\left\{\mathcal{A}\left(f|_{[\tau-1]}\right) = g|_{[\tau-1]}\right\} \cdot \mathbb{P}\left\{\mathcal{M}\left(f|_{[\tau]}, g|_{[\tau-1]}\right) = l_\tau\right\} \cdot \mathbb{P}\left\{\mathcal{C}_{l_\tau}\left(f|_{J_{l_t}}\right) = m_\tau\right\} \\
&\le \mathbb{P}\left\{\mathcal{A}\left(\tilde{f}\Big|_{[\tau-1]}\right) = g|_{[\tau-1]}\right\} \cdot e^{\varepsilon_M}\mathbb{P}\left\{\mathcal{M}\left(\tilde{f}\Big|_{[\tau]}, g|_{[\tau-1]}\right) = l_\tau\right\} \cdot e^{\varepsilon_C}\mathbb{P}\left\{\mathcal{C}_{l_\tau}\left(\tilde{f}\Big|_{J_{l_t}}\right) = m_\tau\right\} \\
&= e^{(\varepsilon_M+\varepsilon_C)} \cdot \mathbb{P}\left\{\mathcal{A}\left(\tilde{f}\Big|_{[\tau]}\right) = g(\tau)\right\}.
\end{aligned}$$

At any time $t > \tau$, since the input data streams are identical, we have,

$$\mathbb{P}\left\{\mathcal{A}\left(f|_{[t]}\right) = g|_{[t]}\right\} \le e^{(\varepsilon_M+\varepsilon_C)} \cdot \mathbb{P}\left\{\mathcal{A}\left(\tilde{f}\Big|_{[t]}\right) = g|_{[t]}\right\}.$$

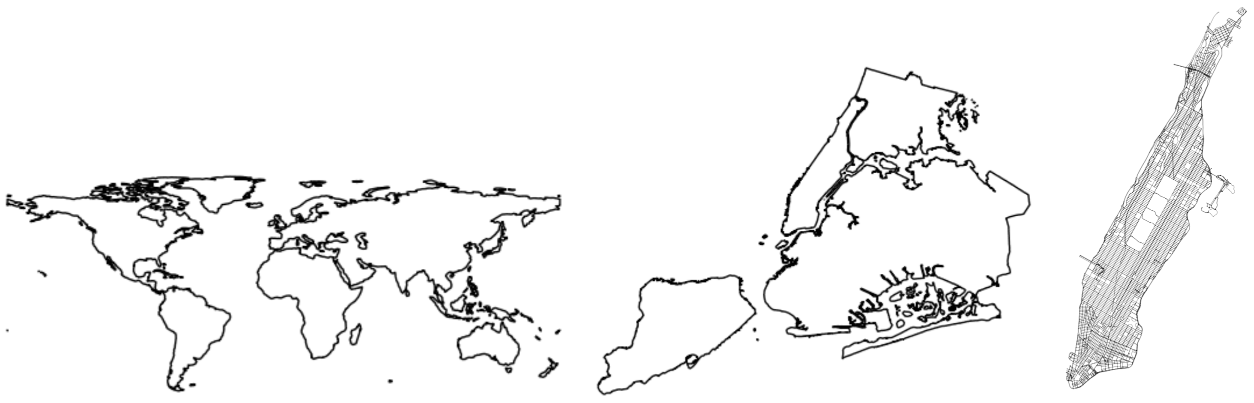
Hence, algorithm $\mathcal{A}$ is $(\varepsilon_M + \varepsilon_C)$-DP. □

Note the following about Algorithm 11 and Theorem 3.5.1: (1) each individual counter can be of any finite dimensionality, (2) we do not assume anything about the relation among the counters, (3) the privacy guarantees are independent of the number of counters, and (4) the algorithm can be easily extended to the case when the input streams to all sub-routines $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$, and $\mathcal{M}$ may be different.

3.5.5. Privacy of PHDStreamTree with counters. We first show that Algorithm 10 is a version of the selective counting algorithm (Algorithm 11). Let $\mathcal{P}(T)$ be a set of all possible subtrees of the tree T. Since each node $v \in T$ is associated with a counter, the collection of counters in $\mathcal{L}(S)$ for any $S \in \mathcal{P}(T)$ is a multi-dimensional counter. Moreover, for any neighboring input streams F and $\tilde{F}$ with $\left\|F - \tilde{F}\right\|_{\triangledown} = 1$, at most one leaf node will differ in the count. Hence $\mathcal{L}(S)$ is a counter with $\varepsilon/2$-DP. $\mathcal{P}(T)$ is thus a set of multi-dimensional counters, each satisfying $\varepsilon/2$-DP. PrivTree$_T$ at time t selects one counter as $\mathcal{L}(T(t))$ from $\mathcal{P}(T)$ and performs counting. Hence, by Theorem 3.5.1, Algorithm 10 is ε-DP.

3.6. Experiments and results

3.6.1. Datasets. We conducted experiments on datasets with the location (latitude and longitude coordinates) of users collected over time. We analyzed the performance of our method on two real-world and three artificially constructed datasets. The two real-world datasets used are Gowalla [14] and NY Taxi [20]. One of the artificially constructed datasets is a version of Gowalla with deletion. The other two artificial datasets both have points on concentric circles, the difference being one has a deletion of points and the other doesn't. We discuss these in detail in the next subsections.



(a) Gowalla

(b) New York taxi (over NY State)

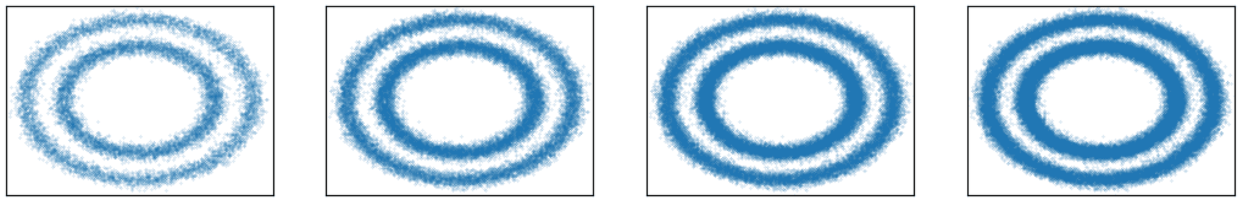
(c) New York Taxi (over road network of Manhattan)

Figure 3.1. Geometry used for datasets

3.6.1.1. *Gowalla.* The Gowalla dataset [14] contains location check-ins by a user along with their user-id and time on the social media app Gowalla. For this dataset, we use the natural land geometry of Earth except for the continent Antarctica (see Figure 3.1a). The size of the dataset after restriction to the geometry is about 6.2 million. We limit the total number of data points to about 200 thousand and follow a daily release frequency based on the available timestamps.

3.6.1.2. *Gowalla with deletion.* To evaluate our method on a dataset with deletion that represents a real-world scenario, we create a version of the Gowalla dataset with deletion. Motivated by the problem of releasing the location of active coronavirus infection cases, we consider each check-in as a report of infection and remove this check-in after 30 days indicating that the reported individual has recovered.

3.6.1.3. *NY Taxi.* The NY Taxi dataset [20] contains the pickup and drop-off locations and times of Yellow Taxis in New York. For this dataset, we only use data from January 2013 which already has more than 14 million data points. We keep the release frequency to 6 hours and restrict total data points to about 10^5 sampling proportional to the number of data points at each time within the overall time horizon. We consider two different underlying geometries for this dataset. The first is the natural land geometry of the NY State as a collection of multiple polygons. The second is the road network of Manhattan Island as a collection of multiple curves. We include a figure of these geometries in Figure 3.1b and Figure 3.1c respectively. The second geometry is motivated by the fact that the majority of taxi pickup locations are on Manhattan Island and also on its road network.


(a) Without deletion, where density on both circles grow with time

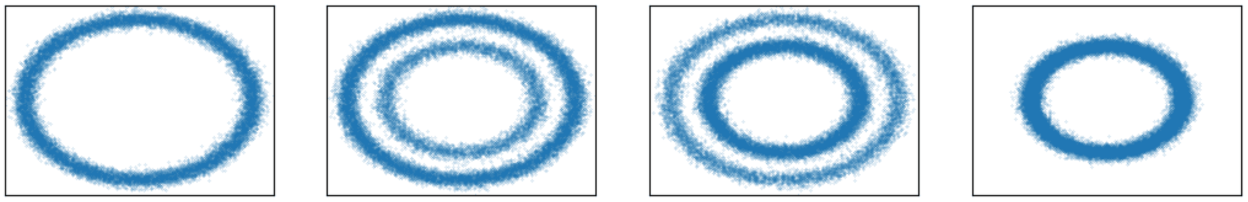
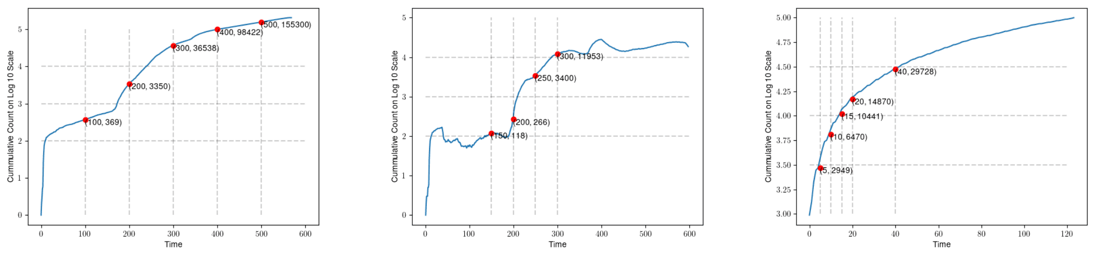

(b) With deletion, where the density gradually changes from predominantly on the outer circle to the inner circle

FIGURE 3.2. Scatter plot for the simulated datasets of two concentric circles showing the resulting location of users at four different times steps progressing from left to right

3.6.1.4. *Concentric circles.* We also conducted experiments on an artificial dataset of points located on two concentric circles. As shown in Figure 3.2, we have two different scenarios of this dataset: (a) where both circles grow over time and (b) where the points first appear in one circle and then gradually move to the other circle. Scenario (b) helps us analyze the performance of our algorithm on a dataset where (unlike Gowalla and NY Taxi datasets) the underlying distribution changes dramatically. We also wanted to explore the performance of our algorithm based on the number of data points it observes in initialization and then at each batch. Thus for this dataset, we first fix a constant batch size for the experiment and then index time accordingly. Additionally, we keep initialization time t_0 as a value in $[0, 1]$ denoting the proportion of total data the algorithm uses for initialization. We explore various parameters on these artificial datasets such as batch size, initialization data ratio, and sensitivity.



(a) Gowalla (b) Gowalla with deletion (c) NY Taxi

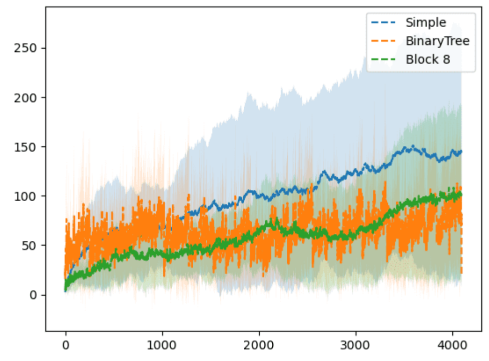
FIGURE 3.3. Cummulative count as time progresses for Gowalla and NY Taxi Dataset. The plot illustrates our motivation for the choice of initialization time.

3.6.2. Initialization time. In Figure 3.3, we show how the total number of data points change with time for the Gowalla and NY Taxi Datasets. For our algorithm to work best, we recommend having points in the order of at least 100 at each time of the differential stream. Hence, based on the cumulative count observed in Figure 3.3 we select minimum $t_0 = 100$ for Gowalla, $t_0 = 150$ for Gowalla with deletion, and $t_0 = 0$ for NY Taxi datasets.

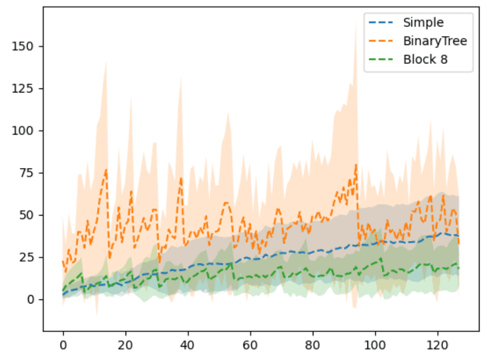
3.6.3. Compute and implementation details. All the experiments were performed on a personal device with 8 cores of 3.60GHz 11th Generation Intel Core i7 processor and 32 GB RAM. Given the compute restriction, for any of the datasets mentioned below, we limit the total number of data points across the entire time horizon to the (fairly large) order of 10^5. Moreover, the maximum time horizon for a stream we have is 300 for Gowalla Dataset. We consider it to be sufficient to show the applicability of our algorithm for various use cases.

We implement all algorithms in Python and in order to use the natural geometry of the data domain we rely on the spatial computing libraries GeoPandas [43], Shapely [31], NetworkX [35], and OSMnx [9]. These libraries allow us to efficiently perform operations such as loading the geometry of a particular known region, filtering points in a geometry, and sampling/interpolating points in a geometry.

3.6.4. Counters. Out of the Simple, Block, and Binary Tree counters (Section 2.4) that were of interest to us, we only use show the results with simple counter here. Figure 3.4 shows an experiment comparing these counters. We observe that the Binary Tree algorithm should be avoided on streams of small time horizons, say $T < 100$. Block counter appears to have lower errors for $T > 30$. However, due to the large growth in error within a block, for $T < 30$ it is unclear if the Block counter is better than the Simple counter. Due to these observations, we restrict our experiments to the Simple and Block counters only. Note that the optimal size of the block in a Block Counter is suggested to be $\lfloor\sqrt{t}\rfloor$ where t is the time horizon for the stream. However, in our method, we do not know in advance the time horizon for the stream that is input for a particular counter. After multiple experiments, we found that a Block counter with size 8 was most effective for our method PHDStream for the datasets in our experiments.



(a) Long time horizon of $2^{12} = 4096$



(b) Short time horizon of $2^7 = 128$

FIGURE 3.4. Comparing Simple, Block with size 8, and Binary Tree counters with privacy budget $\varepsilon = 0.5$. The true data is a random bit stream of 0s and 1s. On the y-axis, we have the absolute error in the value of the counter at any time averaged over 10 independent runs of the counter algorithm.

3.6.5. Initialization. Many real-world datasets such as Gowalla have a slow growth at the beginning of time. In practice, we can hold the processing of such streams until some time $t_0 \in \mathbb{N}$, which we refer to as initialization time, until sufficient data has been collected. Thus for any input data stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ and initialization time $t_0 \in \mathbb{N}$, we use a modified stream $\hat{f} : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ such that $\hat{f}(\cdot, t) = f(\cdot, t + t_0 - 1)$ for all $t \in \mathbb{N}$. We discuss the effects of initialization time t_0 on the algorithm performance in detail in Section 3.6.8

3.6.6. Baselines. Let ε be our privacy budget. Consider the offline synthetic data generation algorithm as per [79], which is to use PrivTree_T with privacy budget $\varepsilon/2$ to obtain a subtree S of T, generate the synthetic count of each node $v \in \mathcal{L}(S)$ by adding Laplace noise with scale $2/\varepsilon$, and sample points in each node $v \in \mathcal{L}(S)$ equal to their synthetic count. Let us denote this algorithm as $\text{PrivTree}_T + \text{Counting}$. A basic approach to convert an offline algorithm to a streaming algorithm is to simply run independent instances of the algorithm at each time $t \in \mathbb{N}$. We presented this idea as a framework in Algorithm 1. Building on the same idea, we create the following three baselines and compare our algorithm with them.

Baseline 1) Offline PrivTree on stream: At any time $t \in \mathbb{N}$, we run an independent version of the offline $\text{PrivTree}_T + \text{Counting}$ on $f(\cdot, t)$, that is the stream data at time t. We want to emphasize that to be differentially private, this baseline requires a privacy parameter that scales with time t and it does NOT satisfy differential privacy with the parameter ε. Thus we expect it to perform much better with an algorithm that is ε-DP. We still use this method as a baseline since, due to a lack of DP algorithms for streaming data, industry applications sometimes naively re-run DP algorithms as more data is collected. However, we show that PHDStream performs competitively close to this baseline.

Baseline 2) Offline PrivTree on differential stream Similar to Baseline 1, at any time $t \in \mathbb{N}$, we run an independent version of the offline $\text{PrivTree}_T + \text{Counting}$ algorithm, but with the input data $\nabla f(\cdot, t)$, that is the differential data observed at t. This baseline satisfies ε-DP. For a fair comparison, at time t, any previous output is discarded and the current synthetic data is scaled by the factor $|f(\cdot, t)|/|\nabla f(\cdot, t)|$.

Baseline 3) Initialization with PrivTree, followed by counting with counters: We first use the offline $\text{PrivTree}_T + \text{Counting}$ algorithm at only the initialization time t_0 and get a subtree S of T as selected by PrivTree. We then create a counter for all nodes $v \in \mathcal{L}(S)$ with privacy budget ε. At any time $t > t_0$, S is not updated, and only the counter for each of the nodes in $\mathcal{L}(S)$ is updated using the differential stream $\nabla f(\cdot, t)$. This baseline also satisfies ε-DP. Note that this algorithm has twice the privacy budget for counting at each time $t > t_0$ as we do not update S. We only have results for this baseline if $t_0 > 0$. We show that PHDStream outperforms this baseline if the underlying density of points changes sufficiently with time.

3.6.7. Performance metric. We evaluate the performance of our algorithm against *range counting queries*, that count the number of points in a subset of the space $\mathcal{X}$. For $\mathcal{X}' \subseteq \mathcal{X}$, the associated range counting query $q_{\mathcal{X}'}$ over a function $h : \mathcal{X} \to \mathbb{R}$ is defined as $q_{\mathcal{X}'}(h) := \sum_{x \in \mathcal{X}'} h(x)$. In our experiments, we use random rectangular subsets of $\mathcal{X}$ as the subregion for a query. Similar to [79], we generate three sets of queries: $10,000$ small, 5000 medium, and 1000 large with area in $[0.01\%, 0.1\%)$, $[0.1\%, 1\%)$, and $[1\%, 10\%)$ of the space $\mathcal{X}$ respectively. We use average relative error over a query set as our metric. At time t, given query set Q, the relative error of a synthetic stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ as compared to the input stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ is thus defined as,

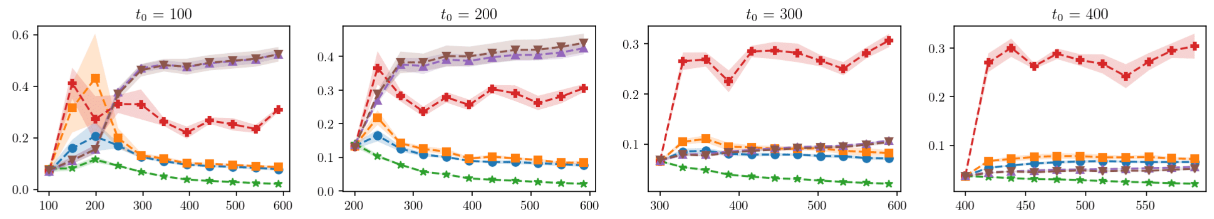
$$r(Q, f, g, t) := \frac{1}{|Q|} \sum_{q \in Q} \frac{\left| q(f(\cdot, t)) - q(g(\cdot, t)) \right|}{\max \left(q(f(\cdot, t)), 0.001 |f(\cdot, t)| \right)}$$

where $0.001|f(\cdot, t)|$ or 1% of $|f(\cdot, t)|$ is a small additive term to avoid division by 0. We evaluate the metric for each query set at a regular time interval and report our findings.

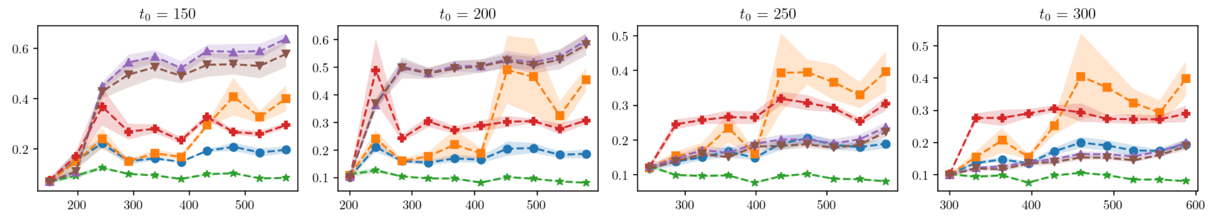
3.6.8. Key results. Our analysis indicates that PHDStream performs well across various datasets and hyper-parameter settings. We include some of the true and synthetic data images in Section 3.7. Due to space constraints, we limit the discussion of the result to a particular setting in Figure 3.5 where we fix the privacy budget to $\varepsilon = 1$, sensitivity to 2, query set to small queries, and explore different values for initialization time t_0. For further discussion, refer to Sections 3.7 and 3.6.9.

PHDStream remains competitive to the challenging non-differentially private Baseline 1 and in almost all cases, it outperforms the differentially private Baseline 2. It also outperforms Baseline 3 if we initialize the algorithm sufficiently early. We discuss above mentioned observations together with the effect of various hyper-parameters below.

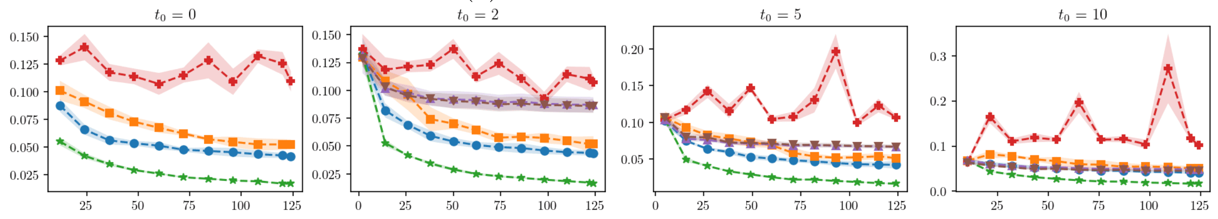
Initialization time: If the dataset grows without changing the underlying distribution too much, it seems redundant to update $T(t)$ with PrivTree$_T$ at each time $t \in \mathbb{N}$. Moreover, when counting



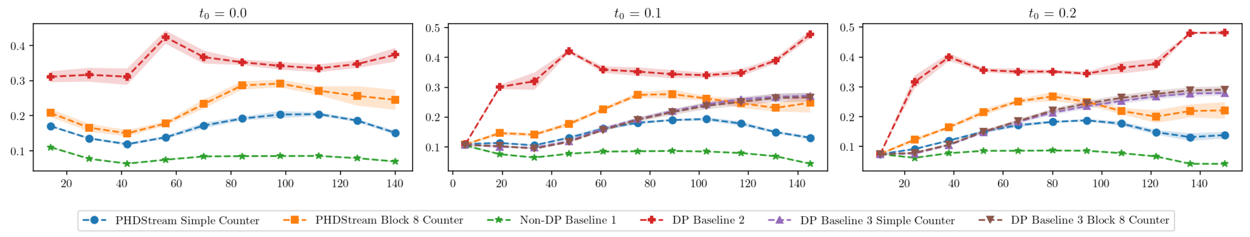
(a) Gowalla



(b) Gowalla with deletion



(c) New York taxi (over NY State)



(d) Concentric circles with deletion

FIGURE 3.5. The progression of relative error in small range queries with time. All experiments are with privacy budget $\varepsilon = 0.5$. Each subplot has a time horizon on the x-axis and corresponds to a particular value of t_0 (increasing from left to right).

using fixed counters, we know that error in a counter grows with time so the performance of the overall algorithm should decrease with time. We observe the same for higher values of initialization time in Figure 3.5. Moreover, we see that Baseline 3, which uses PrivTree only once, outperforms PHDStream for such cases.

Counter type: In almost all cases, for the PHDStream algorithm, the Simple counter has better performance than the Block 8 counter. This can be explained by the fact that for these datasets, the majority of nodes had their counter activated only a few times in the entire algorithm. For example, when using the Gowalla dataset with $t_0 = 100$, on average, at least 90% of total nodes created in the entire run of the algorithm had their counter activated less than 40 times.

Sensitivity and batch size: We explore various values of sensitivity and batch size and the results are as expected - low sensitivity and large batch size improve the algorithm performance. For more details see Section 3.6.9.

3.6.9. Additional results. Figures 3.6, 3.7, 3.8, 3.9, 3.10, and 3.11 show a comparison of performances on different datasets for small-range queries. Each subplot has a time horizon on the x-axis and corresponds to a particular value of privacy budget ε (increasing from left to right) and initialization time t_0 (increasing from top to bottom). We do not show PHDStream with Block counter and Baseline 2 in these figures as they both have very large errors in some cases and that distorts the scale of the figures. We also show some results on medium and large scale range queries (as described in Section 3.6.7) for the dataset Gowalla in Figures 3.12, 3.13 and for the dataset Gowalla with deletion in Figures 3.14, 3.15 respectively.

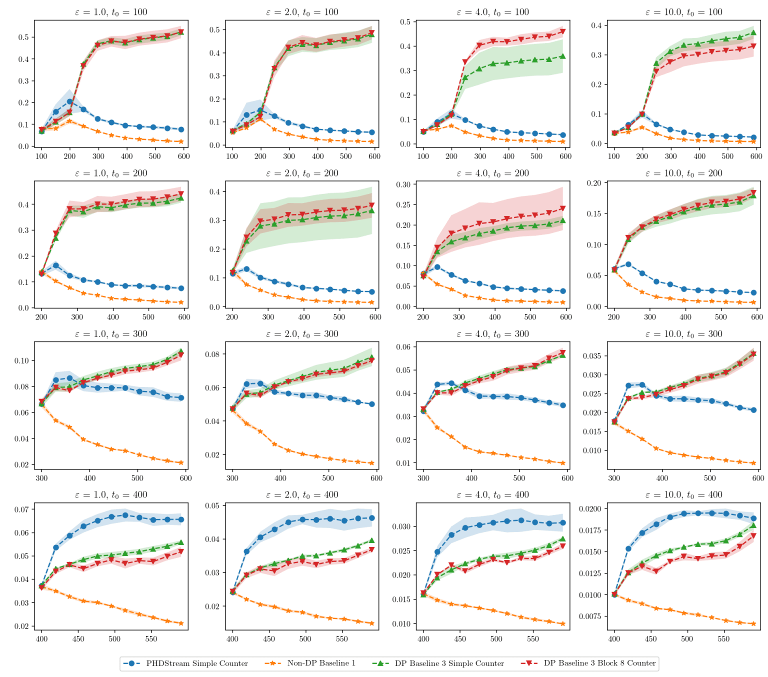


FIGURE 3.6. Query error for PHDStream over the dataset Gowalla and small range queries.

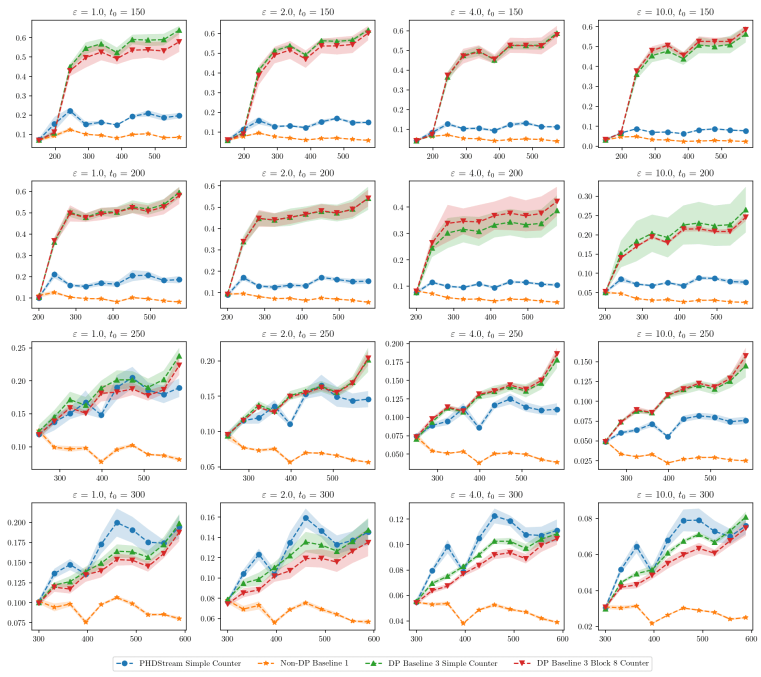


FIGURE 3.7. Query error for PHDStream over the dataset Gowalla with deletion and small range queries.

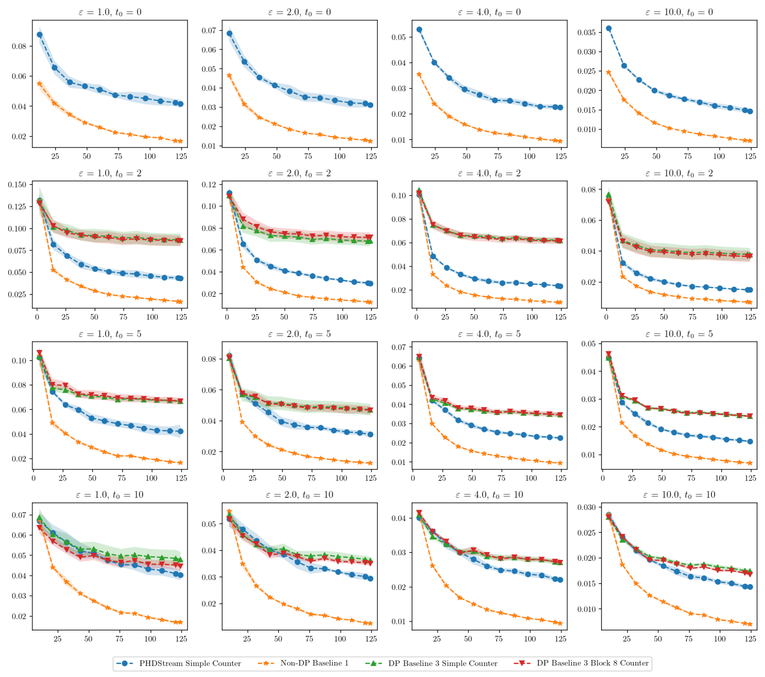


FIGURE 3.8. Query error for PHDStream over the dataset New York (over NY state) and small range queries.

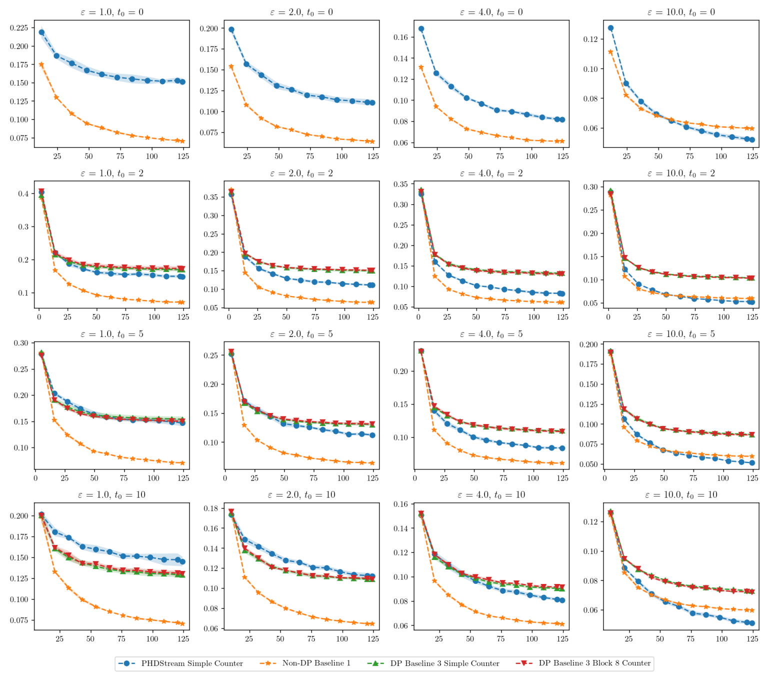


FIGURE 3.9. Query error for PHDStream over the dataset New York (over Manhattan road network) and small range queries.

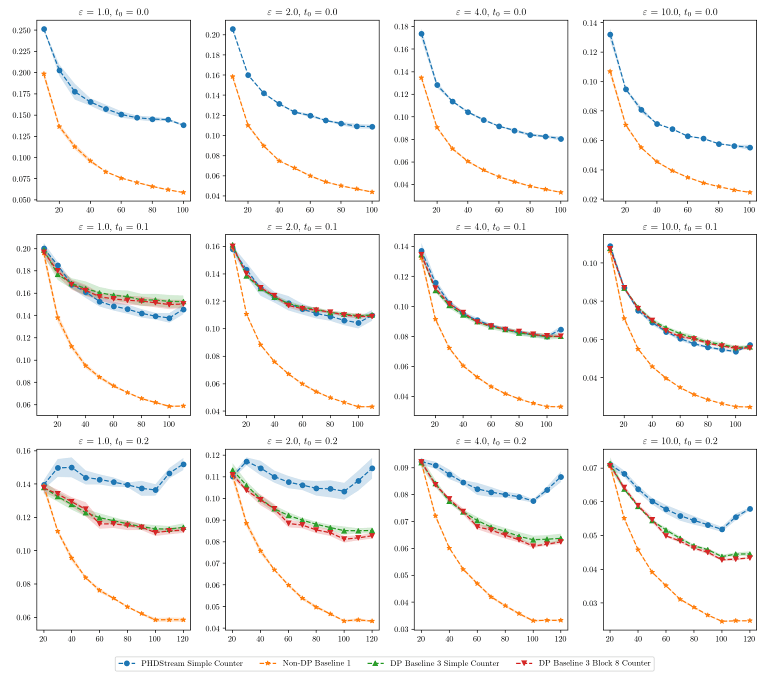


FIGURE 3.10. Query error for PHDStream over the dataset Concentric circles no deletion and small range queries.

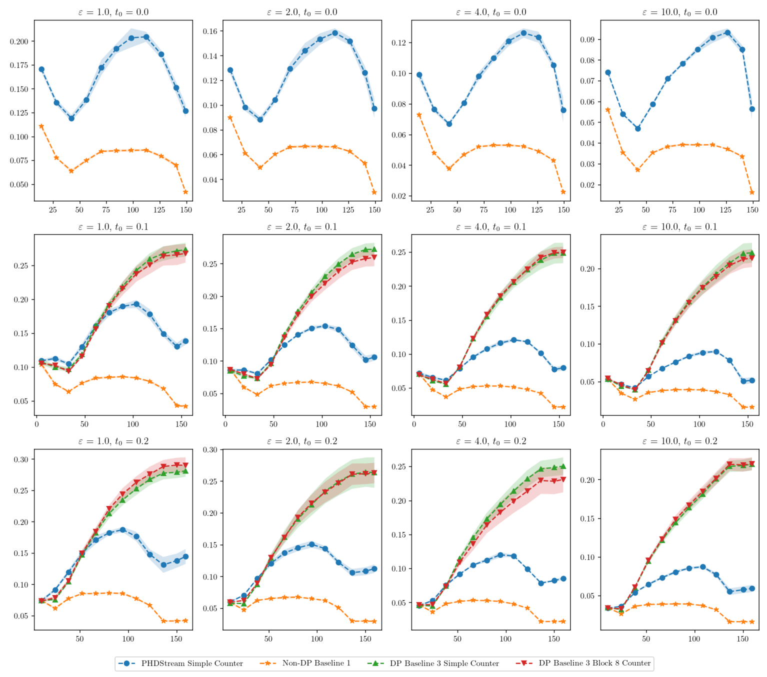


FIGURE 3.11. Query error for PHDStream over the dataset Concentric circles with deletion and small range queries.

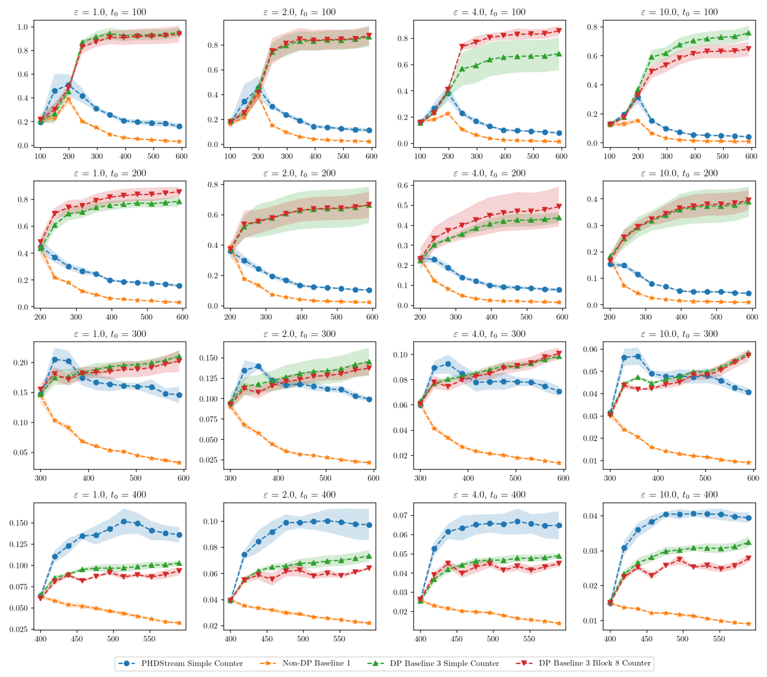


FIGURE 3.12. Query error for PHDStream over the dataset Gowalla and medium range queries.

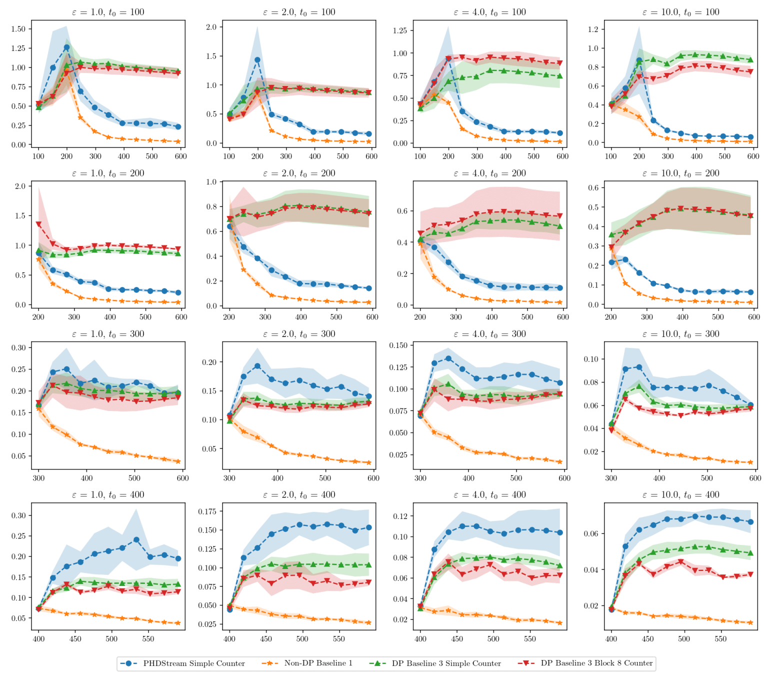


FIGURE 3.13. Query error for PHDStream over the dataset Gowalla and large range queries.

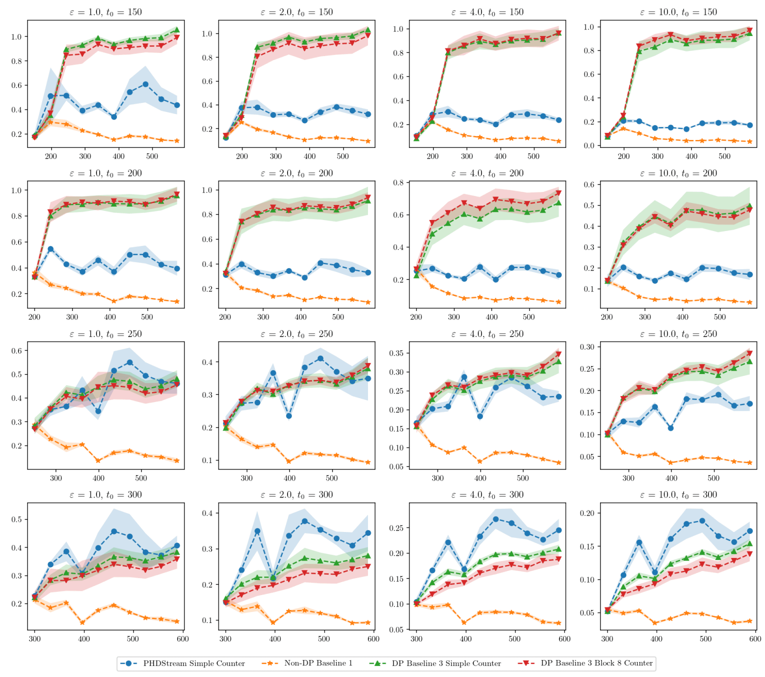


FIGURE 3.14. Query error for PHDStream over the dataset Gowalla with deletion and medium range queries.

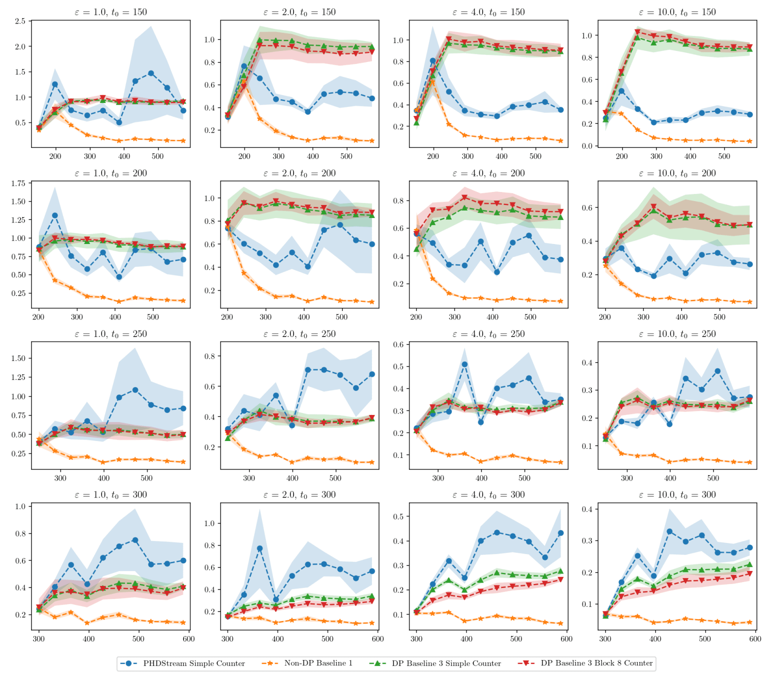


FIGURE 3.15. Query error for PHDStream over the dataset Gowalla with deletion and large range queries.

3.7. Synthetic data scatter plots

In this section, we present a visualization of the synthetic data generated by our algorithm PHDStream with the Simple counter. We present scatter plots comparing the private input data with the generated synthetic data in Figures 3.16, 3.17, 3.18, and 3.19 for the datasets Gowalla, NY Taxi, and Concentric Circles with deletion respectively. Due to space limitations, we only show the datasets a few times, with time increasing from left to right. In each of the figures mentioned above, the first row corresponds to the private input data and is labeled "True". In each subplot, we plot the coordinates present at that particular time after accounting for all additions and deletions so far. Each of the following rows corresponds to one run of the PHDStream algorithm with the Simple counter and for a particular privacy budget ε as labeled on the row. In each subplot, we use black dashed lines to create a partition of the domain corresponding to the leaves of the current subtree as generated by PrivTree_T. For comparison, we overlay the partition created by the algorithm with $\varepsilon = 0.5$ on the true data plot as well.

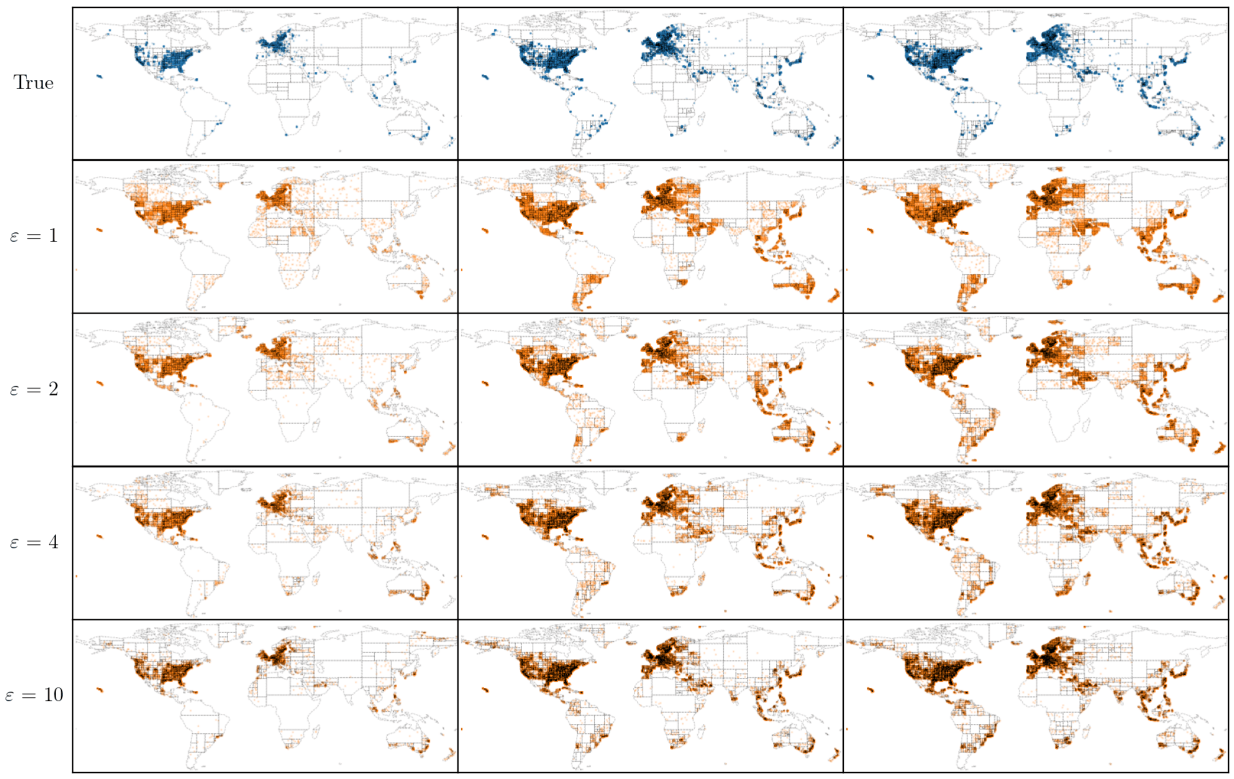


FIGURE 3.16. Scatter plot of True and Synthetic data (generated by PHDStream) over the Gowalla dataset for initialization time index $t_0 = 100$

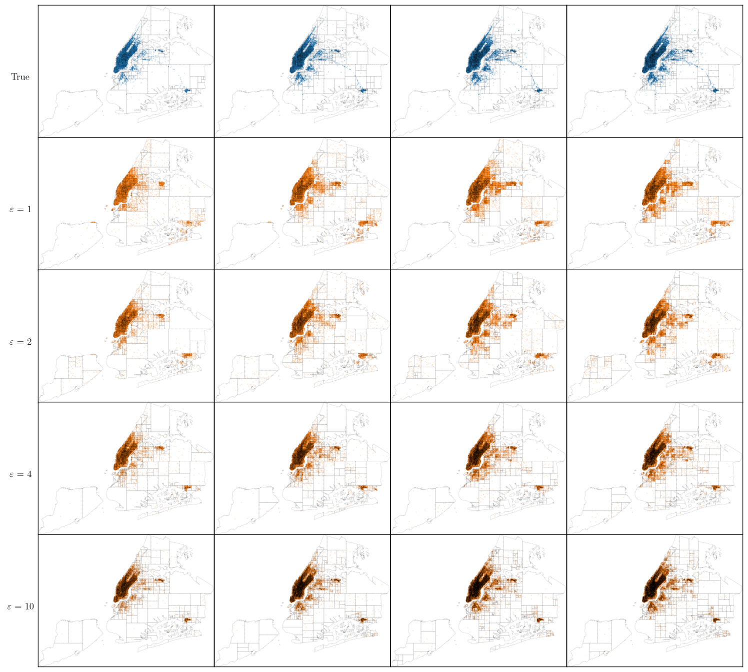


FIGURE 3.17. Scatter plot of True and Synthetic data (generated by PHDStream) over the New York dataset over NY State for initialization time index $t_0 = 2$

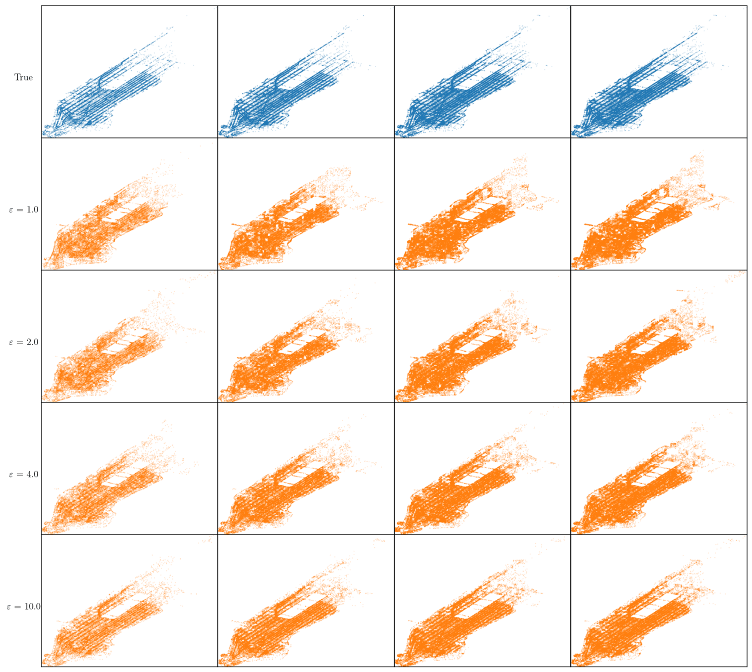


FIGURE 3.18. Scatter plot of True and Synthetic data (generated by PHDStream) over the New York dataset over road network of Manhattan for initialization time index $t_0 = 2$

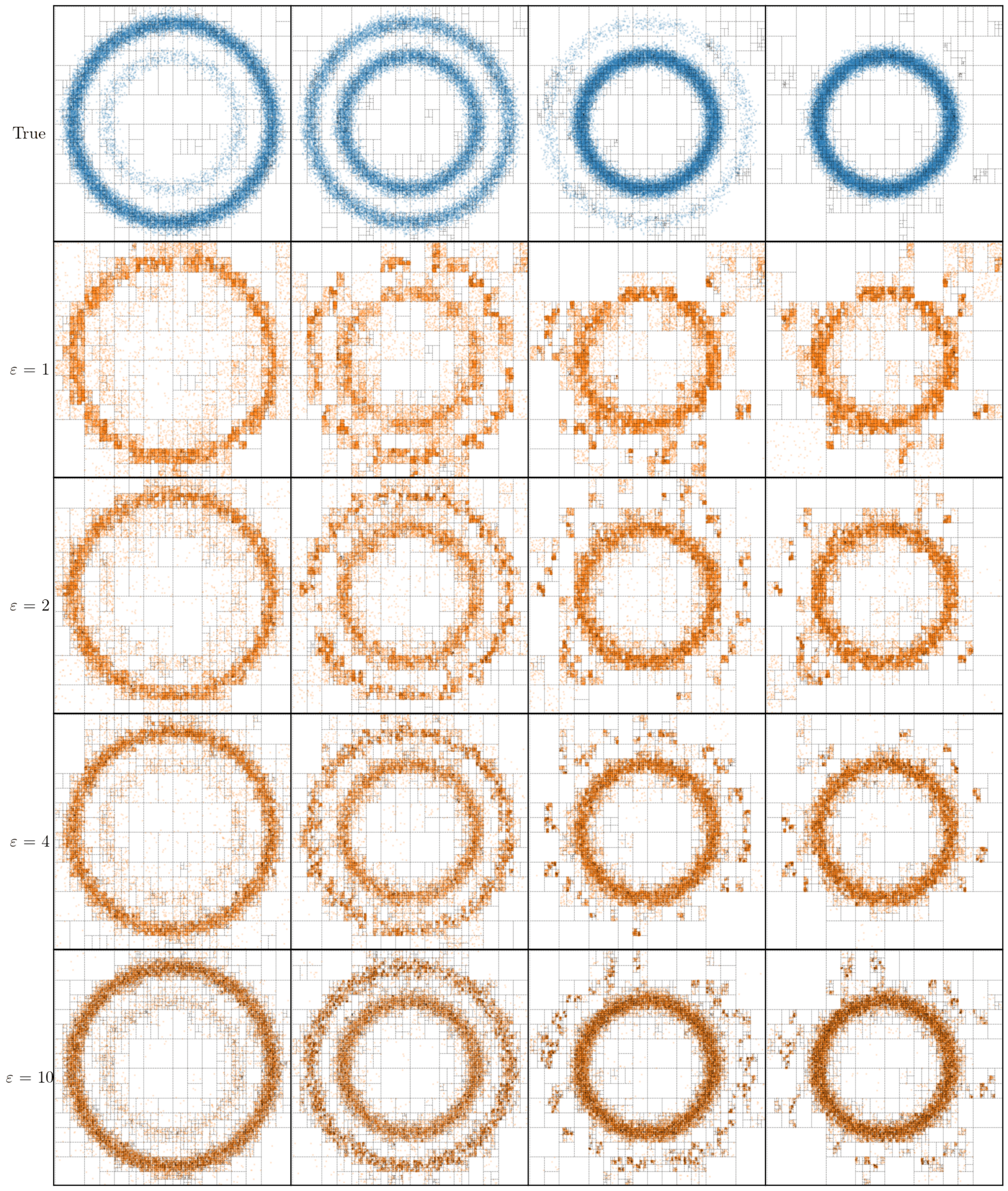


FIGURE 3.19. Scatter plot of True and Synthetic data (generated by PHDStream) over the Concentric circles with deletion dataset for initialization data ratio $t_0 = 0.1$, constant batch size 500

CHAPTER 4

Differentially Private Synthetic Tabular Stream

4.1. Motivation

In Chapter 3 we introduced the problem of multi-dimensional privacy-preserving synthetic data generation. However, we used a method based on a hierarchical decomposition of the space $\mathcal{X}$. While this method works efficiently for low-dimensional spaces, it would not scale well to high-dimensional spaces since the number of nodes will grow exponentially with dimension leading to large time complexity and poor utility.

A typical case of high-dimensional data is *tabular* data, where each record in the dataset consists of (say) p fixed attributes. For example, a dataset about patient records in a hospital, and the census records of a country. Such datasets when freely available to be used by the researchers can be extremely helpful in making informed decisions in domains such as healthcare and public policy. Many existing works have considered the task of differentially private release of synthetic tabular datasets such as [6,29,36,44,54,56,57,58,67,78]. A lot of these works measure the quality of the generated synthetic tabular data using some test functions which typically capture linear statistics of the data. While the task of creating synthetic data for one-time release has been well explored, the task of streaming synthetic data has not. We provide an algorithm for streaming tabular and high-dimensional synthetic data.

4.2. Problem setup

Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_p$ denote a space of p dimensional points such that $\mathcal{X}_i$ is a discrete space of size $|\mathcal{X}_i|$ for any $i \in [p]$. For example, if each data point results from a survey of p boolean questions, then $\mathcal{X} = \{0,1\}^p$. Let $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ be an input stream where $f(x,t)$ denotes the count of point x at time t. We provide an algorithm that generates a privacy-preserving synthetic data stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ that accurately represents the input stream f. We define the term "accurately" rigorously in the subsequent subsections.

Note that here we have assumed that $\mathcal{X}_i$ is discrete for all $i \in [p]$. While this assumption may not hold in practice, we can *discretize* a continuous space by spending some of the privacy budget to create a differentially private histogram and mapping each point to the histogram bin. Some existing works such as [67] have also used PrivTree for creating the histogram.

4.2.1. Marginal queries. In this work, we measure the accuracy of our output stream using marginal queries. A marginal query is a low-dimensional counting query and a formal definition is provided in Definition 4.2.1.

DEFINITION 4.2.1 (k-way marginal query). *A k-way marginal query $q : \mathcal{X} \to \{0, 1\}$ is a mapping defined by a tuple $(c_1, c_2, \ldots, c_k)$ of k column indices and their corresponding values $(v_1, v_2, \ldots, v_k)$ such that $v_i \in \mathcal{X}_{c_i}$ for all $i \in [k]$ and the mapping is defined as,*

$$q(x) = \prod_{i=1}^{k} \left(\mathbb{1}_{\{x_{c_i} = v_i\}} \right), \tag{4.1}$$

for any $x \in \mathcal{X}$.

With a slight abuse of notation, we extend the definition of marginal query from points to datasets in Definition 4.2.2.

DEFINITION 4.2.2 (k-way marginal query for a dataset). *A k-way marginal query q can be extended to any dataset $h : \mathcal{X} \to \mathbb{N}$ as,*

$$q(h) = \sum_{x \in \mathcal{X}} h(x) q(x) \tag{4.2}$$

4.2.2. Accuracy of a one-time algorithm. The quality of an algorithm generating a synthetic dataset is typically measured by the aggregate performance of the generated synthetic data over marginal queries. We define the accuracy formally in Definition 4.2.3.

DEFINITION 4.2.3 (Accuracy of an algorithm for synthetic dataset generation). *Let $\mathcal{A}$ be a randomized algorithm that maps an input dataset $f : \mathcal{X} \to \mathbb{N}$ to a synthetic dataset $g : \mathcal{X} \to \mathbb{N}$. Then, for any $\beta > 0$, $\mathcal{A}$ is said have an accuracy of (α, β), with respect to a set of marginal queries Q, if*

$$\mathbb{P}\left\{ \max_{q \in Q} \left| q(g) - q(f) \right| \geq \alpha \right\} \leq \beta, \tag{4.3}$$

where the probability is taken over the randomness of the algorithm $\mathcal{A}$.

4.2.3. Accuracy of a streaming algorithm for synthetic stream generation. A natural extension of the definition of accuracy from datasets, as presented in Definition 4.2.3, to streams can be created by restricting the stream to any time $t \in \mathbb{N}$ and looking at the accuracy of the dataset present at that time. Definition 4.2.4 builds upon this idea and provides a way to measure the accuracy of streaming algorithms generating synthetic streams.

Definition 4.2.4 (Accuracy of a streaming algorithm). *Let $\mathcal{A}$ be a randomized streaming algorithm that maps an input stream $f : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$ to a synthetic stream $g : \mathcal{X} \times \mathbb{N} \to \mathbb{N}$. Then, at any time $t \in \mathbb{N}$ and for any $\beta > 0$, $\mathcal{A}$ is said have an accuracy of (α, β), with respect to a set of marginal queries Q, if*

$$\mathbb{P}\left\{\max_{q \in Q}\left|q(g_t) - q(f_t)\right| \geq \alpha\right\} \leq \beta, \tag{4.4}$$

where the probability is taken over the randomness of the algorithm $\mathcal{A}$. Here, α may be a function of β and t.

4.3. Offline tabular synthetic dataset generation

Our streaming algorithm uses many ideas from offline algorithms in the literature for dataset generation. Let us first discuss these ideas. Consider the task of generating synthetic tabular data when the dataset is available at once (offline). Let $f : \mathcal{X} \to \mathbb{N}$ be a dataset. Assume we are interested in generating a synthetic dataset $g : \mathcal{X} \to \mathbb{N}$ that is accurate for marginal queries Q. A straightforward approach to generating the synthetic dataset would be: (1) generate a differentially private measurement for all queries using the Laplace Mechanism as

$$m = \left(q(f) + \text{Lap}\left(\frac{\Delta q}{\varepsilon/|Q|}\right)\right)_{q \in Q};$$

(2) find a dataset that minimizes the maximum error over the query set by solving the following optimization problem,

$$\arg\min_{g \in \mathbb{N}^{\mathcal{X}}} \max_{q \in Q}\left|m_q - q(g)\right|. \tag{4.5}$$

There are however two key problems with this approach: (1) the size of the query set is typically polynomial in the dimension p which leads to a very small budget for answering an individual query,

Algorithm 12 Meta algorithm: generating differentially private synthetic tabular dataset

```
1: Input: Given dataset $f$, an ordered set of queries $Q$, privacy budget $\varepsilon$, a differentially private selection mechanism $\mathcal{A}_{Select}$, a subroutine $\mathcal{A}_{Dataset}$ to find a dataset given noisy query measurements, and the number of iterations $k$.
2: Output: A dataset $g \in \mathbb{N}^{\mathcal{X}}$.
3: Create a dataset $h_0 \in \mathbb{N}^{\mathcal{X}}$ with $h_0(x) = 1$ for all $x \in \mathcal{X}$.
4: Set $M \leftarrow \emptyset$ as a set of selected queries and their measurements.
5: for $i = 1, 2, \ldots, k$ do
6:     Set $e_i \leftarrow \left(\left|q(h_{i-1}) - q(f)\right|\right)_{q \in Q}$ as the error in queries.
7:     Select: $l_i \leftarrow \mathcal{A}_{Select}(e_i, 2/\varepsilon)$, an index of query.
8:     Measure: $m_i \leftarrow q_{l_i}(f) + \text{Lap}\left(2\Delta_{q_{l_i}}/\varepsilon\right)$, value of query.
9:     Set $M \leftarrow M \cup \{(q_{l_i}, m_i)\}$; add selected query and its value.
10:    Optimize: Dataset $h_i \leftarrow \mathcal{A}_{Dataset}(M, h_{i-1})$.
```

that is a large amount of noise is added in Laplace Mechanism, and (2) the optimization problem in equation (4.5) is a high-dimensional discrete optimization problem which is NP-Hard and cannot be solved in time polynomial in dimension p. Many existing algorithms thus circumvent the above two problems by: (1) measuring only a subset of the queries in Q which have the largest error, and (2) approximating the optimization problem in Equation 4.5. Let us first assume that we have a way to model the data distribution given the noisy measurements. Let $\mathcal{A}_{Dataset}$ be one such subroutine that takes as input noisy measurements of the queries in Q an initialization dataset (say) $h \in \mathbb{N}^{\mathcal{X}}$, to provide a dataset $g \in \mathbb{N}^{\mathcal{X}}$ that complies with the measurements. In Section 4.3.1, we use $\mathcal{A}_{Dataset}$ as a black-box subroutine and discuss how to iteratively select and measure a subset of the queries. In Section 4.3.2, we then look at some of these methods for creating the dataset given a value of the queries.

4.3.1. The *Select, Measure, Learn, and Iterate* paradigm. Algorithm 12 is a meta-algorithm describing the "select, measure, optimize, and iterate" paradigm. This paradigm is used in several existing methods and has been shown to achieve good empirical accuracy [54]. The algorithm has a fixed number of iterations k. It receives two subroutine algorithms $\mathcal{A}_{Select}$ and $\mathcal{A}_{Dataset}$ which can be treated as a black box for now. $\mathcal{A}_{Select}$ is a differentially private algorithm used for selection, whereas $\mathcal{A}_{Dataset}$ does not guarantee differential privacy and is used to create a dataset based on the values of the queries. Algorithm 12 iteratively produces a series of synthetic datasets $h_1, h_2, \ldots, h_k$ that are, hopefully, more and more closer to the true data f as per the queries Q. In each iteration i, the following happens: (1) using the subroutine $\mathcal{A}_{Select}$, while upholding differential privacy, we select a query q_{l_i} that has the most error on the current synthetic dataset h_{i-1}; (2) an approximation of the value of this query is generated as m_i using the Laplace Mechanism; and finally (3) the dataset is updated from h_{i-1} to h_i by using the sub-routine $\mathcal{A}_{Dataset}$.

4.3.2. Dataset complying with queries. In this subsection, we discuss some algorithms that can be used for $\mathcal{A}_{Dataset}$ in Algorithm 12.

4.3.2.1. *Multiplicative Weights Exponential Mechanism (MWEM).* [36] first introduced the idea of Algorithm 12 and used the *Multiplicative Weights* (MW) algorithm as $\mathcal{A}_{Dataset}$. With the MW algorithm, Step 10 of Algorithm 12 results in a dataset h_i that is $|f|$ times the distribution that satisfies

$$h_i(x) \propto h_{i-1}(x) \cdot \exp\left(q_{l_i}(x) \cdot \frac{m_i - q_{l_i}(h_{i-1})}{2|f|}\right).$$

MWEM solves a convex approximation of Equation (4.5) over the probability simplex in $\mathcal{X}$, we refer the reader to [54] for more details. The algorithm comes with a theoretical guarantee and works quite well for low-dimensional datasets. However, since it requires maintaining a probability distribution over $\mathcal{X}$, it becomes computationally intractable for many real-world datasets.

4.3.2.2. *Probabilistic Graphical Model (PGM).* An alternative to the MW algorithm as $\mathcal{A}_{Dataset}$ in Algorithm 12 is the Probabilistic Graphical Model (PGM) algorithm [58]. PGM further approximates the optimization problem by restricting the solution space from all possible distributions on $\mathcal{X}$ to distributions that can be represented as a graphical model of the form

$$P_\theta(x) = \frac{1}{Z} \exp\left(\sum_{C \in \mathcal{C}} \theta_C(x_C)\right),$$

for all $x \in \mathcal{X}$. Here, $\mathcal{C} \subseteq 2^{[d]}$ is a collection of subsets of $[d]$, θ_C is a function for each $C \in \mathcal{C}$, x_C is the restriction of $x \in \mathcal{X}$ on the column indices in C, and Z is a normalization constant. Thus the model P_θ is defined by low-dimensional functions θ_C, one for each $C \in \mathcal{C}$. The algorithm uses a proximal algorithm to solve the resulting convex optimization problem. PGM has been shown to perform very well in practice and we will be using it in our experiments.

Algorithm 13 Baseline algorithm: StreamingMWEM

```
1: Input: An input data stream $f$, an ordered set of marginal queries $Q$, number of marginals to
   select at any time $k$, the privacy budget $\varepsilon$.
2: Output: A synthetic stream $g$.
3: Initialize $g(0, x) \leftarrow 1$ for all $x \in \mathcal{X}$.
4: for $t = 1, 2, \ldots$ do
5:     Set $I_{t,0} \leftarrow \emptyset$.
6:     for $l = 1, 2, \ldots, k$ do
7:         Set $J_{t,l} \leftarrow [|Q|] \setminus I_{t,l-1}$; as query indices not selected.
8:         Set $e_{t,l} \leftarrow \left(|q_i(\nabla f_t) - q_i(h_{t,l-1})|\right)_{i \in J_{t,l}}$.

9:         // Exponential Mechanism
10:        Sample a query index $\eta_{t,l}$ such that for any $i \in J_{t,l}$,
               $$\mathbb{P}\left\{\eta_{t,l} = i\right\} \propto \exp\left(\frac{\varepsilon}{2k}(e_{t,l})_i\right).$$
11:        Using $j$ as a shorthand for $\eta_{t,l}$.
12:        Set $I_{t,l} \leftarrow I_{t,l-1} \cup \{j\}$.

13:        // Laplace Mechanism
14:        Set $m(t, j) \leftarrow q_j(\nabla f_t) + \mathrm{Lap}\left(\frac{2k}{\varepsilon}\right)$.

15:        // Multiplicative weights
16:        Set $h_{t,l}$ as $|\nabla f_t|$ times the distribution that satisfies
               $$h_{t,l}(x) \propto h_{t,l-1}(x) \cdot \exp\left(q_j(x) \cdot \frac{m_j - q_j(h_{t,l-1})}{2|\nabla f_t|}\right)$$
17:    Set $g_t \leftarrow g_{t-1} + \mathrm{avg}_{l \in [k]}\, h_{t,l}$.
```

4.4. Baseline: Streaming MWEM

We will use Algorithm 1 as our baseline where we simply run an independent version of some offline algorithm on differential data at each time $t \in \mathbb{N}$. As per Theorem 2.3.4, we know that this method satisfies ε-differential privacy.

Next, we discuss the theoretical accuracy of the baseline algorithm. For this discussion, we look at a version of this algorithm by fixing the offline mechanism $\mathcal{A}$ in Algorithm 1 as the MWEM algorithm [**36**] as mentioned in Section 4.3.2.1. Let us refer to the streaming version of MWEM as per Algorithm 1 as *StreamingMWEM* and we present the complete algorithm in Algorithm 13.

Theorem 4.4.1 (Accuacy of StreamingMWEM). *At any time $t \in \mathbb{N}$, StreamingMWEM (Algorithm 13) is $\left(\mathcal{O}\left(|f_t|^{2/3}\left(\frac{(t \log t)\ln|\mathcal{X}|\ln|Q|}{\varepsilon\beta}\right)^{1/3}\right), \beta\right)$ accurate with respect to the set of marginal queries Q.*

PROOF. The below analysis is similar to the analysis of (offline) MWEM algorithm due to [36]. Let us focus the analysis on iteration l of time t.

Selection error: First, we will analyze the error in query selection at Step 9. Let $\text{maxerr}_{t,l}$ denote the maximum possible absolute difference between values of any query in Q as measured on $h_{t,l-1}$ and ∇f_t, that is,

$$\text{maxerr}_{t,l} = \max_{q \in Q} \left| q(h_{t,l-1}) - q(f_t) \right| . \tag{4.6}$$

At the l^{th} iteration at time t, we select the query with index j, where j is a shorthand for $e_{t,l}$. By the utility of exponential mechanism (Theorem 2.2.9) invocated with a privacy budget $\varepsilon/2k$ and sensitivity 1, for any $\beta > 0$, we have,

$$\mathbb{P}\left\{ \left| q_j(h_{t,l-1}) - q_j(\nabla f_t) \right| \leq \text{maxerr}_{t,l} - \frac{4k}{\varepsilon} \ln \frac{|Q|}{\beta} \right\} \leq \beta. \tag{4.7}$$

Additive error: Let us now analyze the error due to the Laplace Mechanism at Step 13. Let $\text{adderr}_{t,l}$ denote the additive error when measuring the query $q_{e_{t,l}}$, that is,

$$\text{adderr}_{t,l} = \left| m_{t,l} - q_{e_{t,l}}(\nabla f_t) \right| . \tag{4.8}$$

Again using j as shorthand for $e_{t,l}$. By concentration of the Laplace random variable (Lemma 2.2.1.1) we have, that for a noise of scale $\frac{2k}{\varepsilon}$,

$$\mathbb{P}\left\{ \left| m_{t,l} - q_{e_{t,l}}(\nabla f_t) \right| > \frac{2k}{\varepsilon} \log \frac{1}{\beta} \right\} = \beta. \tag{4.9}$$

Relative entropy: Similar to [36] we rely on relative entropy to show improvement in each iteration by using the multiplicative weights algorithm. Let the relative entropy at the end of iteration l at time t be given as

$$\Psi_{t,l} = \frac{1}{|\nabla f_t|} \sum_{x \in \mathcal{X}} \nabla f_t(x) \ln \left(\frac{\nabla f_t(x)}{h_{t,l}(x)} \right) . \tag{4.10}$$

Then we have the following relations,

$$\Psi_{t,l} \geq 0, \tag{4.11}$$

$$\Psi_{0,0} \leq ln|\mathcal{X}|\,, \tag{4.12}$$

$$\Psi_{t,l-1} - \Psi_{t,l} \geq \left(\frac{q_{e_{t,l}}(h_{t,l}) - q_{e_{t,l}}(\nabla f_t)}{2|\nabla f_t|}\right)^2 - \left(\frac{m_{t,l} - q_{e_{t,l}}(\nabla f_t)}{2|\nabla f_t|}\right)^2. \tag{4.13}$$

Equation (4.13) can be derived as follows,

$$\begin{aligned}\Psi_{t,l-1} - \Psi_{t,l} &= \frac{1}{|\nabla f_t|}\sum_{x\in\mathcal{X}} \nabla f_t(x) \ln\left(\frac{h_{t,l}(x)}{h_{t,l-1}(x)}\right)\\ &= \frac{1}{|\nabla f_t|}\sum_{x\in\mathcal{X}} \nabla f_t(x) \ln\left(\frac{h_{t,l-1}(x)\cdot \exp\left(q_{e_{t,l}}(x)\cdot\left(\frac{m_{t,l}-q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|}\right)\right)}{h_{t,l-1}(x) Z_{t,l}}\right),\end{aligned}$$

where $Z_{t,l} = \frac{1}{|\nabla f_t|}\sum_{x\in\mathcal{X}} h_{t,l-1}(x) \exp\left(q_{e_{t,l}}(x)\cdot\left(\frac{m_{t,l}-q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|}\right)\right)$ is the normalization constant. So,

$$\begin{aligned}\Psi_{t,l-1} - \Psi_{t,l} &= \frac{1}{|\nabla f_t|}\sum_{x\in\mathcal{X}} \nabla f_t(x)\left(q_{e_{t,l}}(x)\cdot\left(\frac{m_{t,l}-q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|}\right) - \ln Z_{t,l}\right)\\ &= \left(\frac{m_{t,l}-q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|^2}\right) q_{e_{t,l}}(\nabla f_t) - \ln Z_{t,l}.\end{aligned}$$

Using $e^x \leq 1 + x + x^2$ for all $|x| \leq 1$ and $\left| q_{e_{t,l}}(x) \cdot \frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right| \leq 1$, we have,

$$\begin{aligned}
Z_{t,l} &= \frac{1}{|\nabla f_t|} \sum_{x \in \mathcal{X}} h_{t,l-1}(x) \exp\left(q_{e_{t,l}}(x) \cdot \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right) \right) \\
&\leq \frac{1}{|\nabla f_t|} \sum_{x \in \mathcal{X}} h_{t,l-1}(x) \left(1 + q_{e_{t,l}}(x) \cdot \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right) + \left(q_{e_{t,l}}(x) \cdot \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right) \right)^2 \right) \\
&\leq 1 + \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right)^2 + q_{e_{t,l}}(h_{t,l-1}) \cdot \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|^2} \right).
\end{aligned}$$

$$\implies \ln Z_{t,l} \leq \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right)^2 + q_{e_{t,l}}(h_{t,l-1}) \cdot \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|^2} \right).$$

Using this in the entropy difference bound we have,

$$\begin{aligned}
\Psi_{t,l-1} - \Psi_{t,l} &\geq \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|^2} \right) \left(q_{e_{t,l}}(\nabla f_t) - q_{e_{t,l}}(h_{t,l-1}) \right) - \left(\frac{m_{t,l} - q_{e_{t,l}}(h_{t,l-1})}{2|\nabla f_t|} \right)^2 \\
&= \left(\frac{q_{e_{t,l}}(h_{t,l-1}) - q_{e_{t,l}}(\nabla f_t)}{2|\nabla f_t|} \right)^2 - \left(\frac{m_{t,l} - q_{e_{t,l}}(\nabla f_t)}{2|\nabla f_t|} \right)^2. \qquad (4.14)
\end{aligned}$$

Finally: Suppose we are interested in error at time T. Let $\beta > 0$ be the failure probability at some time $t \in [T]$. Then, using Equations (4.7), (4.9) and (4.13) and a union bound over $l \in [k]$, with probability at least $1 - \beta$, for all $l \in [k]$ simultaneously,

$$\text{maxerr}_{t,l} \leq \left| q_{e_{t,l}}(h_{t,l-1}) - q_{e_{t,l}}(\nabla f_t) \right| + \frac{4k}{\varepsilon} \log\left(\frac{2k|Q|}{\beta} \right), \qquad (4.15)$$

and,

$$\text{adderr}_{t,l} = \left| m_{t,l} - q_{e_{t,l}}(\nabla f_t) \right| \leq \frac{2k}{\varepsilon} \log\left(\frac{2k}{\beta} \right). \qquad (4.16)$$

Combining the above two equations with Equation 4.13 we have, that with probability at least $1 - \beta$,

$$\text{maxerr}_{t,l} \leq \left(4|f_t|^2 \left(\Psi_{t,l-1} - \Psi_{t,l} \right) + \text{adderr}_{t,l}^2 \right)^{1/2} + \frac{4k}{\varepsilon} \log\left(\frac{2k|Q|}{\beta} \right).$$

Finally, we can bound the maximum error in approximating the differential dataset as,

$$\begin{aligned}
\max_{q\in Q}\left|q(\nabla g_t) - q(\nabla f_t)\right| &= \max_{q\in Q}\left|q\left(\operatorname{avg}_{l\in[k]} h_{t,l}\right) - q(\nabla f_t)\right| \\
&\le \operatorname{avg}_{l\in[k]} \max_{q\in Q}\left|q(h_{t,l}) - q(\nabla f_t)\right| = \operatorname{avg}_{l\in[k]} \mathrm{maxerr}_{t,l} \\
&\le \operatorname{avg}_{l\in[k]} \left(4|\nabla f_t|^2\left(\Psi_{t,l-1} - \Psi_{t,l}\right) + \mathrm{adderr}_{t,l}^2\right)^{1/2} + \frac{4k}{\varepsilon}\log\left(\frac{2k|Q|}{\beta}\right) \\
&= \left(\frac{4|\nabla f_t|^2}{k}\left(\Psi_{t,0} - \Psi_{t,k}\right) + \mathrm{adderr}_{t,l}^2\right)^{1/2} + \frac{4k}{\varepsilon}\log\left(\frac{2k|Q|}{\beta}\right) \\
&\le \left(\frac{4|\nabla f_t|^2}{k}\ln|\mathcal{X}| + \mathrm{adderr}_{t,l}^2\right)^{1/2} + \frac{4k}{\varepsilon}\log\left(\frac{2k|Q|}{\beta}\right) \\
&\le 2|\nabla f_t|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \frac{2k}{\varepsilon}\log\left(\frac{2k}{\beta}\right) + \frac{4k}{\varepsilon}\log\left(\frac{2k|Q|}{\beta}\right).
\end{aligned}$$

Let us suppose we are interested in error at time $T \in \mathbb{N}$. Taking a union bound over time we have, that with probability at least $1-\beta$, for all $t \le T$ simultaneously,

$$\begin{aligned}
\max_{q\in Q}\left|q(g_t) - q(f_t)\right| &\le \sum_{t=1}^{T} \max_{q\in Q}\left|q(\nabla g_t) - q(\nabla f_t)\right| \\
&\le \sum_{t=1}^{T}\left(2|\nabla f_t|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \frac{2k}{\varepsilon}\log\left(\frac{2kt}{\beta}\right) + \frac{4k}{\varepsilon}\log\left(\frac{2kt|Q|}{\beta}\right)\right) \\
&\le 2|f_T|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \frac{2k}{\varepsilon}\sum_{t=1}^{T}\left(\log\left(\frac{2t|Q|}{\beta}\right) + 2\log\left(\frac{2t|Q|^2}{\beta}\right)\right) \\
&\le 2|f_T|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \frac{6k}{\varepsilon}\sum_{t=1}^{T}\left(\log\left(\frac{2t|Q|^{5/3}}{\beta}\right)\right) \\
&\le 2|f_T|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \frac{6kT}{\varepsilon}\log\left(\frac{2T|Q|^{5/3}}{\beta}\right).
\end{aligned}$$

Let us compare the upper bound to a function of the form $u(k) = \frac{a}{\sqrt{k}} + bk$, then we can optimize for the value of k with $k_* = \left(\frac{a}{2b}\right)^{2/3}$. This results in $u(k_*) = \left(2^{1/3} + 2^{-1/3}\right)a^{2/3}b^{1/3}$. Using this optimal value in our upper bound so far, we have,

$$\max_{q\in Q}\left|q(g_t)-q(f_t)\right| \le \mathcal{O}\left(\left(|f_T|\sqrt{\ln|\mathcal{X}|}\right)^{2/3}\left(\frac{T}{\varepsilon}\log\left(\frac{T|Q|^{5/3}}{\beta}\right)\right)^{1/3}\right)$$
$$\le \mathcal{O}\left(|f_T|^{2/3}\left(\frac{\ln|\mathcal{X}|\ln|Q|\,(T\log T)}{\varepsilon\beta}\right)^{1/3}\right).$$

□

4.5. Main Algorithm

4.5.1. Outline. In a nutshell, our algorithm also follows the "select, measure, optimize, and iterate" paradigm described in Algorithm 12. At any time $t \in \mathbb{N}$, the goal is to ensure that f_t and g_t are close to each other as evaluated using the queries Q. We start with a dataset $h_{t,0} = g_{t-1}$ and update it over k iterations from $h_{t,0}, h_{t,1}, \ldots,$ to $h_{t,k}$. At any iteration $l \in [k]$, we select a query index $\eta_{t,l} \in [|Q|]$ for which our dataset $h_{t,l-1}$ has approximately the highest error when compared to f_t. We will discuss how exactly this selection is done soon, but for now, let us accept it as a black-box. At the end of the k iterations, g_t is set to some aggregate of the datasets $h_{t,1}, h_{t,2}, \ldots,$ and $h_{t,k}$. We present our proposed method as a meta-algorithm in Algorithm 14.

4.5.2. Measure. Let $m : \mathbb{N} \times [|Q|] \to \mathbb{R}$ be a map such that $m(t,i)$ denotes our differentially private approximation of $q_i(f_t)$, that is the value of query $q_i \in Q$ at time t. Since a single query may be selected at multiple time instances, we use a counter algorithm to measure the value of the query efficiently over time. We associate each query in Q with an instance of some counter Algorithm, say $\mathcal{A}_{Counter}$. Consider a query $q_i \in Q$ and let C_i be its corresponding counter. We use the notation $C_i(t)$ to conveniently refer to the value of the counter C_i at time t. Let $N_i(t) \subseteq [t]$ be the time instances until time t when the query q_i was selected to be measured using the true data. Also, let $\overline{N}_i(t) \coloneqq [t] \setminus N_i(t)$ be the time instances until time t at which query q_i was not selected. Then the output $C_i(t)$ of the counter algorithm is based solely on the stream $\nabla f_{N_i(t)}$.
However, to generate the dataset g_t we need an approximate measurement of the value $q_i(f_t)$. In other words, we are missing the measurement of the query on times $\overline{N}_i(t)$ when the index i was not selected. At any such time $\tau \in \overline{N}_i(t)$, since q_i was not selected, we assume that the query value $q_i(g_\tau)$ on the synthetic dataset g_τ is close to the true value $q_i(f_\tau)$. We create a map $r : \mathbb{N} \times [|Q|] \to \mathbb{R}$

Algorithm 14 Main algorithm: streaming differentially private synthetic tabular stream

```
1: **Input:** An input data stream $f$, an ordered set of marginal queries $Q$, number of marginals to select at any time $k$, the privacy budget $\varepsilon$, a counter algorithm $\mathcal{A}_{Counter}$, and a subroutine $\mathcal{A}_{Dataset}$ to find a dataset given noisy query measurements.
2: **Output:** A synthetic stream $g$.
3: Initialize $C_1, C_2, \ldots, C_{|Q|}$ as independent instances of the counter algorithm $\mathcal{A}_{Counter}$, one for each query in the set $Q$, with privacy budget $\varepsilon/2k$.
4: Initialize $g(0, x) \leftarrow 1$ for all $x \in \mathcal{X}$.
5: Initialize $m(0, i) \leftarrow 0$ for all $i \in [|Q|]$; query measurements of selected queries
6: Initialize $r(0, i) \leftarrow 0$ for all $i \in [|Q|]$; remainder of query value for times when the query is not selected.
7: **for** $t = 1, 2, \ldots$ **do**
8:     Set $I_{t,0} \leftarrow \emptyset$.
9:     **for** $l = 1, 2, \ldots, k$ **do**
10:        Set $J_{t,l} \leftarrow [|Q|] \setminus I_{t,l-1}$; as query indices not selected.
11:        Set $e_{t,l} \leftarrow \big(|q_i(\nabla f_t + g_{t-1}) - q_i(h_{t,l-1})|\big)_{i \in J_{t,l}}$.
12:        $\eta_{t,l} \leftarrow ExponentialMechanism\big(e_{t,l}, \varepsilon/2k\big)$.
13:        Using shorthand $j$ for $\eta_{t,l}$.
14:        Set $I_{t,l} \leftarrow I_{t,l-1} \cup \{j\}$.
15:        Update counter $C_j$ by counting on $\nabla f_t$.
16:        Set $r(t, j) \leftarrow r(t-1, j)$.
17:        Set $m(t, j) \leftarrow C_j + r(t, j)$.
18:        Set $h_{t,l} \leftarrow \mathcal{A}_{Dataset}\left(\Big\{\big(q_i, m(t,i)\big)\Big\}_{i \in I_{t,l}}, h_{t,l-1}\right)$.
19:    Set $g_t \leftarrow \mathrm{avg}_{l \in [k]}\, h_{t,l}$.
20:    Set $C_i(t) \leftarrow C_i(t-1)$ for all $i \in [|Q|] \setminus I_{t,k}$.
21:    Set $r(t, i) \leftarrow q_i(g_t) - C_i(t)$ for all $i \in [|Q|] \setminus I_{t,k}$.
```

such that $r(t,i)$ denotes our differentially private approximation of the value of query q_i over times in $\overline{N}_i(t)$. Assuming $r(0,i) = 0$, we define $r(t,i)$ for any $t \in \mathbb{N}$ as,

$$r(t,i) = \begin{cases} q_i(g_t) - C_i(t), & t \in N_i(t), \\ r(t-1, i), & \textit{otherwise}. \end{cases}$$

Finally, our differentially private approximation $m(t,i)$ of the query q_i at time t becomes

$$m(t,i) = C_i(t) + r(t,i).$$

4.5.3. Optimize. At any time t and iteration l, Algorithm 14 uses the Algorithm $\mathcal{A}_{Dataset}$ as a subroutine to generate the synthetic dataset $h_{t,l}$ using the query indices selected so far at time t, that is $\{\eta_{t,1}, \dots, \eta_{t,l}\}$, and their corresponding differentially private values $\{m(t, \eta_{t,1}), \dots, m(t, \eta_{t,l})\}$. $\mathcal{A}_{Dataset}$ can be any algorithm and is not required to satisfy differential privacy.

4.5.4. Select. We are finally ready to talk about query selection. During iteration l of time t, we want to select the query with maximum error over the synthetic dataset $h_{t,l-1}$ as compared to the true dataset f_t. However, accessing $q(f_t)$ results in high sensitivity. Indeed a simple change at some time $\tau \in \mathbb{N}$ can affect the selection at all times $t > \tau$.
To control the sensitivity, we follow the same trick as in Algorithm 6 and approximate f_t as $g_{t-1} + \nabla f_t$ for selection. For any query $q_i \in Q$, $l \in [k]$, and $t \in \mathbb{N}$, we define

$$e_{t,l} \coloneqq \left(|q_i(\nabla f_t + g_{t-1}) - q_i(h_{t,l-1})|\right)_{i\in[|Q|]} .$$

Finally, we use the Exponential Mecnahism as defined in Definition 2.2.7 for selecting a query index $\eta_{t,l}$ given the vector of query utilities $e_{t,l}$. Note that Algorithm 14 does not find the error for all queries but instead only for queries that have not been chosen so far at iteration l of time t (whose indices are in the set $J_{t,l}$).

Lemma 4.5.0.1 (Privacy of Algorithm 14). *If the algorithm $\mathcal{A}_{Counter}$ satisfies ε-differential privacy, then Algorithm 14 satisfies ε-differential privacy.*

Proof. Note that Algorithm 14 is an instance of Algorithm 11 from Chapter 3. Theorem 3.5.1 implies that Algorithm 11 is differentially private. Hence, Algorithm 14 also satisfies differential privacy. Moreover, since we split the budget as $\varepsilon/2$ for the selection (Exponential Mechanism) and $\varepsilon/2$ for counters at any time t, Algorithm 14 satisfies ε-differential privacy. □

4.6. Accuracy analysis

In this section, we try to find a bound on the accuracy of Algorithm 14. The analysis mostly follows what we did in Theorem 4.4.1 and we use the notations $\mathrm{maxerr}_{t,l}$, $\mathrm{adderr}_{t,l}$ and $\Psi_{t,l}$ from the proof of that theorem. Additionally, we use the notation maxerr_t to denote the maximum error comparing the input and synthetic stream snapshot at time t, that is

$$\mathrm{maxerr}_t \coloneqq \max_{q\in Q} \left| q(g_t - f_t)) \right| , \tag{4.17}$$

note that there is only one index in the subscript here, unlike $\mathrm{maxerr}_{t,l}$.

Selection error: In Algorithm 14, we use an approximation of the true data in the exponential mechanism, such that at any time $t \in \mathbb{N}$, we use $\nabla f_t + g_{t-1}$ instead of f_t for true data. This introduces bias which can be analyzed as,

$$\begin{aligned}
\max_{q\in Q}\left|q(h_{t,l-1}) - q(\nabla f_t - g_{t-1})\right| &= \max_{q\in Q}\left|q(h_{t,l-1} - f_t) - q(g_{t-1} - f_{t-1}))\right| \\
&\geq \max_{q\in Q}\left|\left|q(h_{t,l-1} - f_t)\right| - \left|q(g_{t-1} - f_{t-1}))\right|\right| \\
&\geq \max_{q\in Q}\left|q(h_{t,l-1} - f_t)\right| - \max_{q\in Q}\left|q(g_{t-1} - f_{t-1}))\right| \\
&= \mathrm{maxerr}_{t,l} - \mathrm{maxerr}_{t-1}\,.
\end{aligned}$$

At the l^{th} iteration at time t, we select the query with index i, where i is a shorthand for $e_{t,l}$. By the utility of the Exponential mechanism (Theorem 2.2.9) invocated with a privacy budget $\varepsilon/2k$ and sensitivity 1, we have,

$$\begin{aligned}
\mathbb{P}\left\{\left|q_i(h_{t,l-1}) - q_i(\nabla f_t + g_{t-1})\right| \leq \max_{q\in Q}\left|q(h_{t,l-1}) - q(\nabla f_t + g_{t-1})\right| - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} &\leq \beta \\
\mathbb{P}\left\{\left|q_i(h_{t,l-1}) - q_i(\nabla f_t + g_{t-1})\right| \leq \mathrm{maxerr}_{t,l} - \mathrm{maxerr}_{t-1} - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} &\leq \beta \\
\mathbb{P}\left\{\left|q_i(h_{t,l-1} - f_t) + q_i(f_{t-1} - g_{t-1})\right| \leq \mathrm{maxerr}_{t,l} - \mathrm{maxerr}_{t-1} - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} &\leq \beta \\
\mathbb{P}\left\{\left|q_i(h_{t,l-1} - f_t)\right| + \max_{q\in Q}\left|q(f_{t-1} - g_{t-1})\right| \leq \mathrm{maxerr}_{t,l} - \mathrm{maxerr}_{t-1} - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} &\leq \beta \\
\mathbb{P}\left\{\left|q_i(h_{t,l-1} - f_t)\right| + \mathrm{maxerr}_{t-1} \leq \mathrm{maxerr}_{t,l} - \mathrm{maxerr}_{t-1} - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} &\leq \beta,
\end{aligned}$$

which results in,

$$\mathbb{P}\left\{\left|q_i(h_{t,l-1} - f_t)\right| \leq \mathrm{maxerr}_{t,l} - 2\,\mathrm{maxerr}_{t-1} - \frac{4k}{\varepsilon}\ln\frac{|Q|}{\beta}\right\} \leq \beta. \tag{4.18}$$

Additive error: Using $m_{t,l} = C_i(t) + r_i(t)$, additive error becomes,

$$\mathrm{adderr}_{t,l} = \left|C_i(t) + r_i(t) - q_i(f_t)\right|. \tag{4.19}$$

Let $N_i(t) \subseteq N$ be the times when query index i is selected by the exponential mechanism at or before time t. Let $\overline{N}_i(t) = [t] \setminus N_i(t)$. Then,

$$\begin{aligned}
\mathrm{adderr}_{t,l} &\leq \left|C_i(t) - q_i\left(f_{N_i(t)}\right)\right| + \left|r_i(t) - q_i\left(f_{\overline{N}_i(t)}\right)\right| \\
&\leq \left|C_i(t) - q_i\left(f_{N_i(t)}\right)\right| + \left|q_i(g_{t-1}) - C_i(t-1) - q_i\left(f_{\overline{N}_i(t)}\right)\right| \\
&\leq \left|C_i(t) - q_i\left(f_{N_i(t)}\right)\right| + \left|q_i(g_{t-1}) - C_i(t-1) - q_i\left(f_{\overline{N}_i(t)}\right)\right| \\
&\leq \left|C_i(t) - q_i\left(f_{N_i(t)}\right)\right| + \left|q_i(g_{t-1}) - q_i(f_{t-1})\right| + \left|C_i(t-1) - q_i\left(f_{N_i(t-1)}\right)\right|
\end{aligned}$$

By the utility guarantees of the binary tree mechanism we have,

$$\mathbb{P}\left\{\left|C_i(t) - q_i\left(f_{N_i(t)}\right)\right| \geq \frac{c}{\varepsilon}\ln\left(\frac{1}{\beta}\right)\left(\ln\left|N_i(t)\right|\right)^{3/2}\right\} \leq \beta, \tag{4.20}$$

for some constant c.

Overall: Then, using a union bound and Equations (4.18) and (4.20), with probability at least $1-\beta$, for all $l \in [k]$ and $t \in [T]$ simultaneously,

$$\mathrm{maxerr}_{t,l} \leq \left|q_i(h_{t,l-1} - f_t)\right| + 2\,\mathrm{maxerr}_{t-1} + \frac{4k}{\varepsilon}\ln\left(\frac{2kT|Q|}{\beta}\right), \tag{4.21}$$

and

$$\mathrm{adderr}_{t,l} \leq \frac{c}{\varepsilon}\ln\left(\frac{2kT}{\beta}\right)\left(\left(\ln\left|N_i(t)\right|\right)^{3/2} + \left(\ln\left|N_i(t-1)\right|\right)^{3/2}\right) + \mathrm{maxerr}_{t-1}\,. \tag{4.22}$$

Relative entropy Recall that Equation (4.13) from the proof of Theorem 4.4.1 states,

$$\Psi_{t,l-1} - \Psi_{t,l} \geq \left(\frac{q_{e_{t,l}}(h_{t,l}) - q_{e_{t,l}}(f_t)}{2|f_t|}\right)^2 - \left(\frac{m_{t,l} - q_{e_{t,l}}(f_t)}{2|f_t|}\right)^2.$$

Combining the equations we have,

$$\mathrm{maxerr}_{t,l} \leq \left(4|f_t|^2\left(\Psi_{t,l-1} - \Psi_{t,l}\right) + \mathrm{adderr}_{t,l}^2\right)^{1/2} + 2\,\mathrm{maxerr}_{t-1} + \frac{4k}{\varepsilon}\ln\left(\frac{2kT|Q|}{\beta}\right).$$

Then, similar to the proof of Theorem 4.4.1, for any $t \in [T]$, we have,

$$\max_{q\in Q}\left|q(g_t) - q(f_t)\right| \leq 2|f_t|\sqrt{\frac{\ln|\mathcal{X}|}{k}} + \mathrm{adderr}_{t,l} + 2\,\mathrm{maxerr}_{t-1} + \frac{4k}{\varepsilon}\ln\left(\frac{2kT|Q|}{\beta}\right).$$

This results in the following recursive relation

$$\max_{q\in Q}\left|q(g_t)-q(f_t)\right| \le 2|f_t|\sqrt{\frac{\ln|\mathcal{X}|}{k}}+\frac{2c}{\varepsilon}\ln\left(\frac{2kT}{\beta}\right)(\ln t)^{3/2}+\frac{4k}{\varepsilon}\ln\left(\frac{2kT|Q|}{\beta}\right)+3\,\mathrm{maxerr}_{t-1}\,. \tag{4.23}$$

We can simplify the above recursive relation such that at time T we have,

$$\max_{q\in Q}\left|q(g_T)-q(f_T)\right| \le \sum_{t=1}^{T}3^{T-t}\left(2|f_t|\sqrt{\frac{\ln|\mathcal{X}|}{k}}+\frac{2c}{\varepsilon}\ln\left(\frac{2kT}{\beta}\right)(\ln t)^{3/2}+\frac{4k}{\varepsilon}\ln\left(\frac{2kT|Q|}{\beta}\right)\right).$$

Note that the above bound has a term exponential in time and thus is not better than what we had for the accuracy of StreamingMWEM in Theorem 4.4.1. However, the results of our empirical experiments (Section 4.7.4) suggest that the proposed method outperforms the StreamingMWEM. The reason that we do not have a better bound in the theoretical proof is that our algorithm depends on the output of the previous time step, which results in the worst-case recursive relation as mentioned in Equation (4.23).

4.7. Experiments and results

We explore the performance of our algorithm on real-world datasets. We use Algorithm 14 with Probabilistic Graphical Model (PGM) [58] as the subroutine $\mathcal{A}_{Dataset}$. We briefly introduced PGM in Section 4.3.2.2. Similar to the experiments in Section 3.6.8 of Chapter 3, we found that the Simple counter works the best for our use case as we do not have a very large time horizon for the stream. So, for the experiments in this section, we use the Simple counter as subroutine $\mathcal{A}_{Counter}$. In the subsequent subsections, we discuss the details of over experiments.

4.7.1. Datasets.

4.7.1.1. *Eviction.* The Eviction Dataset [1] contains eviction notices filed with the San Francisco Rent Board from January 1, 1997. The dataset has an attribute “File Date” which represents the date on which the eviction notice was filed with the Rent Board of Arbitration. We use the value of this attribute to construct our time index for the stream. We fix a synthetic data release frequency as weekly or bi-weekly, and based on the attribute File Date and this frequency, create the time index for our data. In the results, we refer to the corresponding streams as *Eviction-weekly* and *Eviction-bi-weekly* respectively. We limit the dataset to 3 location-based categorical attributes - “Eviction Notice Source Zipcode”, “Supervisor District”, and “Neighborhoods”, and all binary attributes such as - “Non Payment”, “Breach”, and “Illegal Use”. Thus the domain of the dataset used was 22 dimensional with 19 binary and 3 categorical attributes.

4.7.1.2. *Adult.* Adult dataset [8] has been used in many previous works in this domain and so we also use this dataset. We use a processed version of the dataset released in the source code provided by [54]. Note that there is no notion of time in this dataset. We artificially create time in two ways which results in the following two streams: (1) *Adult-randomized*: we fix a constant batch size say B, that is the number of points that are added at each time, then at any time t, we simply add B points to our stream that are sampled uniformly at random from the Adult dataset without replacement; (2) *Adult-ordered*: the stream is also created by adding a fixed batch of B points, except the points are selected deterministically, where we first sort the entire dataset in increasing order and then add the next B points based on the indices at any time. We explore a small and large value of batch size B as 50 and 200 respectively. Note that the data stream Adult-ordered is interesting in the sense that the query values may change very drastically over time.

4.7.2. Workload of queries. In the experiments, we aim to preserve all 2-way marginals on the space $\mathcal{X}$. However, instead of using the set of all 2-way marginals queries as Q, we use the set of all 2-way *workloads*.

DEFINITION 4.7.1 (Workload). *A k-way workload W is defined by a tuple of k column indices, $(c_1, c_2, \ldots, c_k)$ such that $c_i \in [p]$ for all $i \in [k]$ and $c_1 < c_2 < \ldots < c_k$. Let columns$(W) = (c_1, c_2, \ldots, c_k)$. Then, W is an ordered collection of all k-way marginal queries on columns(W), where the order is taken as the lexicographic order of the values corresponding to queries. Furthermore, we denote the number of marginal queries in W with $|W|$. The value of a workload over a dataset $f : \mathcal{X} \to \mathbb{N}$ is defined as the tuple $W(f) = \big(q(f)\big)_{q \in W}$.*

Using workloads instead of marginal queries is a common practice in the literature and is sometimes referred to as the *marginal trick* [54]. To see the advantage, note that the collections of queries in a workload create a disjoint space in the sense that a user can contribute data in at most one of them. Thus we can use the Parallel composition of differential privacy (Theorem 2.1.4) and do not have to divide the privacy budget among the queries in a workload when estimating them. In other words, estimating any workload is a histogram query with sensitivity 1 for any pair of neighboring datasets. This is true even though there are (likely) multiple queries in the workload.

Note that since we are now using workloads instead of marginal queries, the following changes are needed in Algorithm 14 to ensure compatibility,

(1) Q is an ordered set of workloads such that for any $i \in |Q|$, $|q_i|$ denotes the number of marginal queries in q_i;

(2) each counter instance C_i, corresponding to the workload q_i, is a multi-dimensional counter of dimension $|q_i|$, as defined in Section 3.5.2 of Chapter 3;

(3) at an iteration l of time t, we define $e_{t,l}$ as

$$e_{t,l} = \left(\frac{1}{|q_i|} \left\| q_i(\nabla f_t + g_{t-1}) - q_i(h_{t,l-1}) \right\|_{\ell_1} - \left| \mathcal{X}_{columns(q_i)} \right| \right)_{i \in J_{t,l}},$$

where $\left| \mathcal{X}_{columns(q_i)} \right|$ is a bias correction term accounting for the number of queries in a workload;

(4) for 2-way workloads, the resulting sensitivity of the exponential mechanism is $\frac{1}{4}$.

4.7.3. Evaluation metrics. We measure the performance of our algorithm using the error introduced by the generated synthetic data in answering queries. Since we use workloads in our algorithm, we measure the error in queries grouped by the workloads. This results in two levels of aggregation, one for queries within a workload and the second over different workloads.

Let us assume that we are looking for error at time t for the synthetic stream g given input stream f and set of workloads Q. Then, we define the workload errors as,

(1) *Workload error*: For any workload W, we define workload error at time t as the average error in queries within W, that is

$$workloadError(W, f, g, t) := \frac{1}{|W|} \sum_{q \in W} \left| q(f(\cdot, t)) - q(g(\cdot, t)) \right|.$$

(2) *Relative workload error*: Similar to workload error, for any workload W, we define relative workload error at time t as the average relative error in queries within W, that is

$$relativeWorkloadError(W, f, g, t) := \frac{1}{|W|} \sum_{q \in W} \left| \frac{q(f(\cdot, t)) - q(g(\cdot, t))}{q(f(\cdot, t))} \right|.$$

Our final metric is the aggregated (average and maximum) error over all workloads. Given a set Q containing workloads, an input stream f, and a synthetic stream g, at any time t we define:

(1) *Average over workload errors (AvgWE)*: as the average workload error over workloads in Q, that is

$$AvgWE(Q, f, g, t) := \frac{1}{|Q|} \sum_{W \in Q} workloadError(W, f, g, t).$$

(2) *Maximum over workload errors (MaxWE)*: as the maximum workload error over workloads in Q, that is

$$MaxWE(Q, f, g, t) := \max_{W \in Q} \left(workloadError(W, f, g, t) \right).$$

(3) *Average over relative workload errors (AvgRelWE)*: as the average relative workload error over workloads in Q, that is

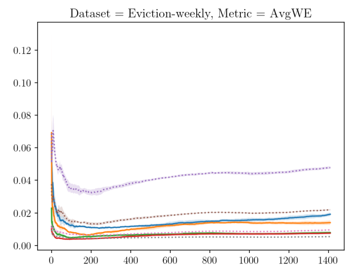
$$AvgRelWE(Q, f, g, t) := \frac{1}{|Q|} \sum_{W \in Q} relativeWorkloadError(W, f, g, t).$$

(4) *Maximum over relative workload errors (MaxRelWE)*: as the maximum relative workload error over workloads in Q, that is

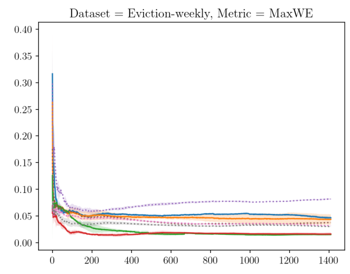
$$MaxRelWE(Q, f, g, t) := \max_{W \in Q} \left(relativeWorkloadError(W, f, g, t) \right).$$

4.7.4. Results. Figures 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 provide our results for the Eviction-weekly, Eviction-bi-weekly, Adult-randomized-bs-50, Adult-randomized-bs-200, Adult-ordered-bs-50, and Adult-ordered-bs-200 datasets respectively. The horizontal axis in all of these figures represents time and the vertical axis is the metric mentioned in the caption of the corresponding subfigure. At the beginning of time, we see a large variance in the metrics, and the proposed method has a larger error in some experiments. However, as time passes, in most cases, our method outperforms the baseline across various datasets, metrics, and privacy budgets. Moreover, we observe that among the various metrics, the most variation occurs in metrics that measure the worst-case errors: MaxWE and MaxRelWE.

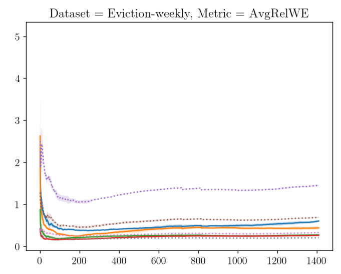
We also provide some metrics as a tabular view in Tables 4.1, 4.2, and 4.3 for Eviction, Adult-randomized, and Adult-ordered datasets to facilitate the comparison. For various datasets and privacy budgets ε, we show the value of the metric for synthetic data generated by the baseline and the proposed methods. We present the value of each metric averaged over the last 10 time steps.



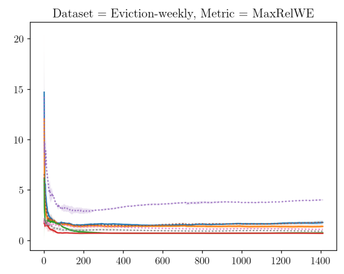
(a) Average of workload errors



(b) Maximum of workload errors

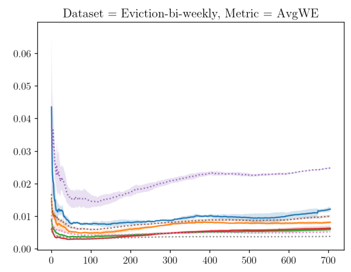


(c) Average of relative workload errors

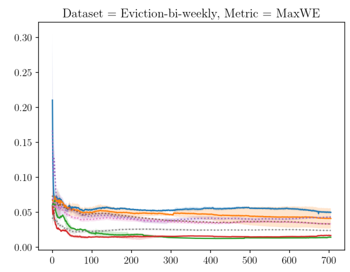


(d) Maximum of relative workload errors

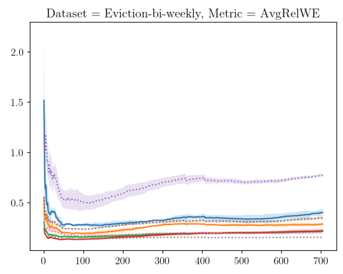
FIGURE 4.1. Metrics over time for the Eviction-weekly dataset



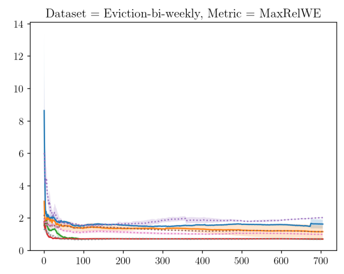
(a) Average of workload errors



(b) Maximum of workload errors

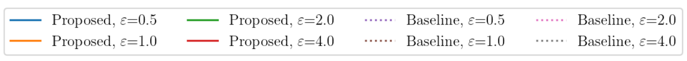


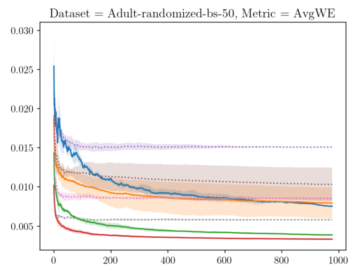
(c) Average of relative workload errors



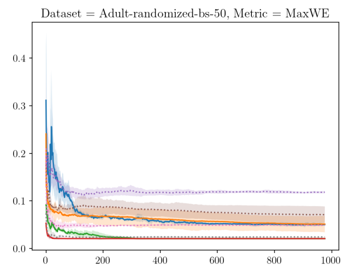
(d) Maximum of relative workload errors

FIGURE 4.2. Metrics over time for the Eviction-bi-weekly dataset

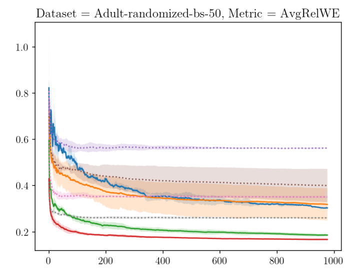




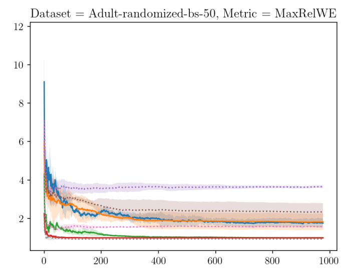
(a) Average of workload errors



(b) Maximum of workload errors

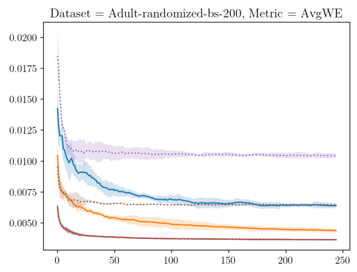


(c) Average of relative workload errors

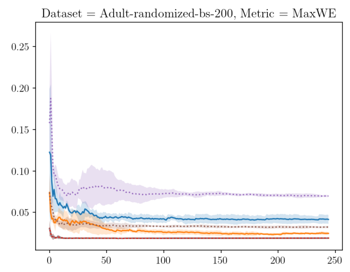


(d) Maximum of relative workload errors

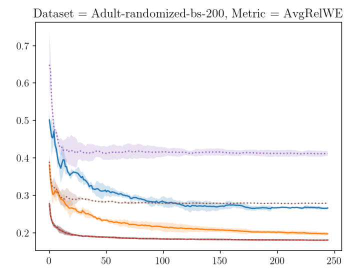
FIGURE 4.3. Metrics over time for the Adult-randomized-bs-50 dataset



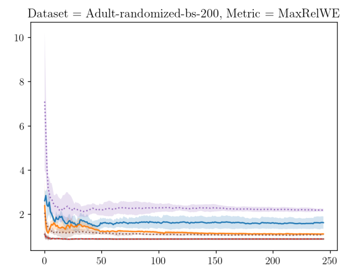
(a) Average of workload errors



(b) Maximum of workload errors

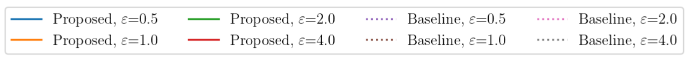


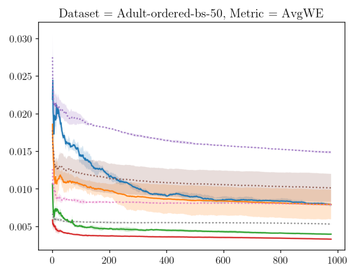
(c) Average of relative workload errors



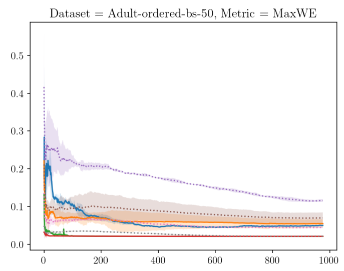
(d) Maximum of relative workload errors

FIGURE 4.4. Metrics over time for the Adult-randomized-bs-200 dataset

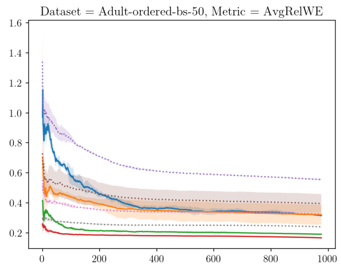




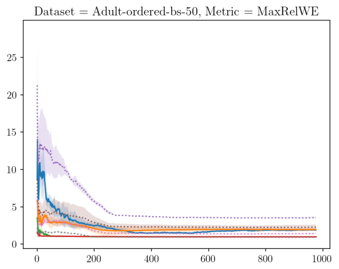
(a) Average of workload errors



(b) Maximum of workload errors

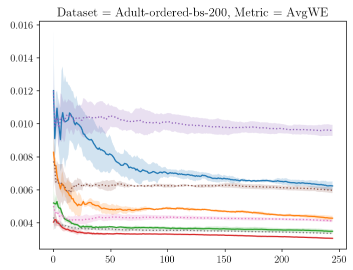


(c) Average of relative workload errors

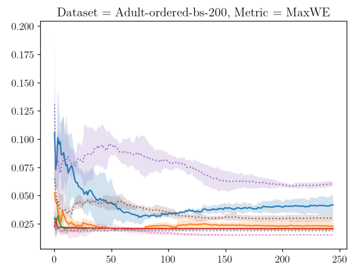


(d) Maximum of relative workload errors

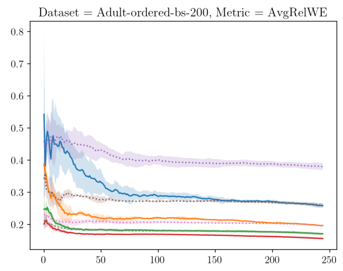
FIGURE 4.5. Metrics over time for the Adult-ordered-bs-50 dataset



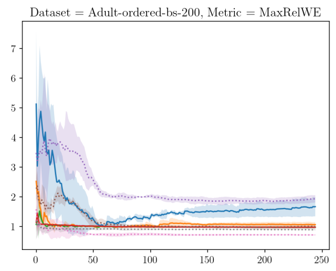
(a) Average of workload errors



(b) Maximum of workload errors



(c) Average of relative workload errors



(d) Maximum of relative workload errors

FIGURE 4.6. Metrics over time for the Adult-ordered-bs-200 dataset

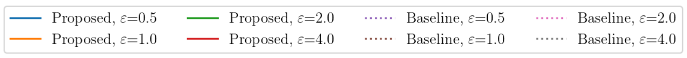


Table 4.1. Query error metrics for the baseline and proposed methods for the *Eviction* dataset across various privacy budgets.

Dataset	ε	**Metric**	**Baseline**	**Proposed**
Eviction-bi-weekly	0.5	AvgRelWE	0.7719	**0.4009**
		AvgWE	0.0248	**0.0121**
		MaxRelWE	2.0291	**1.6396**
		MaxWE	**0.043**	0.0499
	1.0	AvgRelWE	0.3487	**0.2864**
		AvgWE	0.01	**0.0082**
		MaxRelWE	**1.1644**	1.1651
		MaxWE	**0.0331**	0.0411
	2.0	AvgRelWE	**0.2046**	0.2166
		AvgWE	**0.0053**	0.006
		MaxRelWE	1.0008	**0.71**
		MaxWE	0.0333	**0.0141**
	4.0	AvgRelWE	**0.1571**	0.2207
		AvgWE	**0.0039**	0.0064
		MaxRelWE	**0.6945**	0.71
		MaxWE	0.0238	**0.0167**
Eviction-weekly	0.5	AvgRelWE	1.4523	**0.6024**
		AvgWE	0.0478	**0.0191**
		MaxRelWE	4.0261	**1.7817**
		MaxWE	0.0818	**0.0467**
	1.0	AvgRelWE	0.6906	**0.4409**
		AvgWE	0.0219	**0.014**
		MaxRelWE	1.7988	**1.3978**
		MaxWE	**0.0378**	0.0436
	2.0	AvgRelWE	0.3309	**0.2746**
		AvgWE	0.0096	**0.0079**
		MaxRelWE	1.1009	**0.7573**
		MaxWE	0.032	**0.0154**
	4.0	AvgRelWE	**0.2066**	0.2632
		AvgWE	**0.0054**	0.0075
		MaxRelWE	0.923	**0.71**
		MaxWE	0.0303	**0.0162**

Table 4.2. Query error metrics for the baseline and proposed methods for the *Adult-randomized* dataset across various privacy budgets.

Adult-randomized-bs-200	0.5	AvgRelWE	0.4113	**0.2658**
		AvgWE	0.0104	**0.0064**
		MaxRelWE	2.1987	**1.6246**
		MaxWE	0.0698	**0.0419**
	1.0	AvgRelWE	0.2786	**0.1975**
		AvgWE	0.0065	**0.0044**
		MaxRelWE	**1.0789**	1.1166
		MaxWE	0.0323	**0.0249**
	2.0	AvgRelWE	0.1802	**0.1802**
		AvgWE	0.0036	**0.0036**
		MaxRelWE	0.8907	**0.8907**
		MaxWE	0.0191	**0.0191**
	4.0	AvgRelWE	0.1802	**0.1802**
		AvgWE	0.0036	**0.0036**
		MaxRelWE	0.8907	**0.8907**
		MaxWE	0.0191	**0.0191**
Adult-randomized-bs-50	0.5	AvgRelWE	0.5624	**0.3035**
		AvgWE	0.0151	**0.0075**
		MaxRelWE	3.6469	**1.7575**
		MaxWE	0.1182	**0.0504**
	1.0	AvgRelWE	0.4005	**0.3191**
		AvgWE	0.0103	**0.0079**
		MaxRelWE	2.3292	**1.8162**
		MaxWE	0.0712	**0.0514**
	2.0	AvgRelWE	0.351	**0.1863**
		AvgWE	0.0085	**0.0039**
		MaxRelWE	1.5646	**0.977**
		MaxWE	0.0488	**0.0208**
	4.0	AvgRelWE	0.2606	**0.1673**
		AvgWE	0.0058	**0.0033**
		MaxRelWE	**0.9187**	0.9778
		MaxWE	0.0243	**0.0208**

TABLE 4.3. Query error metrics for the baseline and proposed methods for the *Adult-ordered* dataset across various privacy budgets.

Dataset	ε	**Metric**	**Baseline**	**Proposed**
Adult-ordered-bs-200	0.5	AvgRelWE	0.381	**0.2593**
		AvgWE	0.0096	**0.0063**
		MaxRelWE	1.9203	**1.6522**
		MaxWE	0.0601	**0.0413**
	1.0	AvgRelWE	0.2608	**0.1972**
		AvgWE	0.006	**0.0043**
		MaxRelWE	**1.006**	1.067
		MaxWE	0.0302	**0.0232**
	2.0	AvgRelWE	0.1969	**0.1711**
		AvgWE	0.0041	**0.0035**
		MaxRelWE	**0.7189**	0.9752
		MaxWE	**0.0152**	0.0208
	4.0	AvgRelWE	0.1705	**0.1563**
		AvgWE	0.0034	**0.0031**
		MaxRelWE	**0.8939**	0.9776
		MaxWE	**0.019**	0.0208
Adult-ordered-bs-50	0.5	AvgRelWE	0.5559	**0.3165**
		AvgWE	0.0149	**0.008**
		MaxRelWE	3.5589	**1.9028**
		MaxWE	0.1157	**0.0497**
	1.0	AvgRelWE	0.3953	**0.3214**
		AvgWE	0.0102	**0.008**
		MaxRelWE	2.2663	**1.9528**
		MaxWE	0.0695	**0.0547**
	2.0	AvgRelWE	0.3264	**0.1904**
		AvgWE	0.0079	**0.004**
		MaxRelWE	1.4234	**0.9798**
		MaxWE	0.0444	**0.0208**
	4.0	AvgRelWE	0.2402	**0.1667**
		AvgWE	0.0053	**0.0033**
		MaxRelWE	**0.9192**	0.978
		MaxWE	0.0211	**0.0208**

4.8. A new (unbounded) Block counter

We extend the Two-Level counter mechanism (also referred to as Block counter) due to [12] to unbounded streams. We present it formally in Algorithm 15. The idea is similar to how the bounded Binary Mechanism is extended to the unbounded Hybrid Mechanism in [12]. As shown in [12], an optimal block size of the Bounded Block Counter for a stream of size T is $\sqrt{T}$. The key idea is to partition the time dimension of the stream $f : \mathbb{N} \to \mathbb{R}$ into intervals of size $4, 9, 16, \ldots$ (that is perfect squares), and within each of the corresponding intervals, we use a bounded block counter of block size $2, 3, 4, \ldots$ respectively.

Algorithm 15 Unbounded Block Counter

```
1: **Input:** An input data stream $f : \mathbb{N} \to \mathbb{R}$, the privacy budget $\varepsilon$.
2: **Output:** A synthetic stream $g : \mathbb{N} \to \mathbb{R}$.
3: Initialize partition size $T \leftarrow 4$.
4: Initialize block size $B \leftarrow 2$.
5: Last block value $\alpha_{lastBlock} \leftarrow 0$.
6: True value within block $\alpha_{trueInBlock} \leftarrow 0$.
7: Synthetic value within block $\alpha_{synthInBlock} \leftarrow 0$.
8: Time when the last partition changed $t_{atPartition} \leftarrow 0$.
9: Set $g(0) = 0$.
10: **for** $t = 1, 2, \ldots$ **do**
11:     Set $\delta \leftarrow t - t_{atPartition}$.
12:     Update $\alpha_{trueInBlock} \leftarrow \alpha_{trueInBlock} + f(t)$.
13:     **if** $\delta = kB$ for some $k \in Z$ **then**
14:         Update $\alpha_{lastBlock} \leftarrow \alpha_{lastBlock} + \alpha_{tueInBlock} + \mathrm{Lap}\left(\frac{2}{\varepsilon}\right)$.
15:         Update $\alpha_{trueInBlock} \leftarrow 0$ and $\alpha_{synthInBlock} \leftarrow 0$.
16:         Set $g(t) \leftarrow \alpha_{lastBlock}$.
17:         **if** $\delta = T$ **then**
18:             Update $t_{atPartition} \leftarrow t$.
19:             Update $B \leftarrow B + 1$ and $T \leftarrow B^2$.
20:     **else**
21:         Update $\alpha_{synthInBlock} \leftarrow \alpha_{synthInBlock} + f(t) + \mathrm{Lap}\left(\frac{2}{\varepsilon}\right)$.
22:         Set $g(t) \leftarrow \alpha_{lastBlock} + \alpha_{synthInBlock}$.
23:     Release $g(t)$.
```

THEOREM 4.8.1 (Privacy of unbounded block counter). *The unbounded block counter, as presented in Algorithm 15, satisfies ε-differential privacy.*

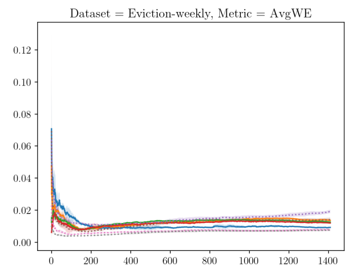
PROOF. Note that Algorithm 15 is exactly the block counter algorithm, except the size of the block changes over time. However, the change in the block size is independent of the input data stream. Hence, similar to the Block counter, Algorithm 15 is ε-differentially private. □

4.8.1. Results. In this section, we present our results for empirical analysis of the proposed unbounded block counter (Algorithm 15) as compared to the simple counter (Algorithm 2), when used as the subroutine $\mathcal{A}_{Dataset}$ in Algorithm 14. We refer to the unbounded block counter simply as the block counter.

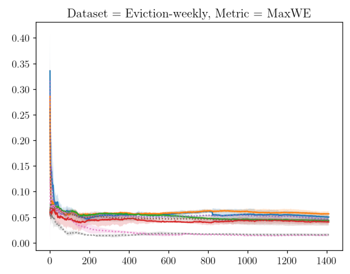
We know that the block counter performs better than the simple counter only after sufficiently large time t, we only use the datasets Eviction-weekly, Adult-ordered-bs-50, and Adult-randomized-bs-50. The length of time horizons for these datasets are 1409, 977, and 977 respectively.

Similar to Section 4.7.4, we present the findings of our experiments as Figures 4.7, 4.9, and 4.8 which show various error metrics over time. We also present a tabular view of these metrics in Tables 4.4 and 4.5 for Eviction and Adult datasets respectively.

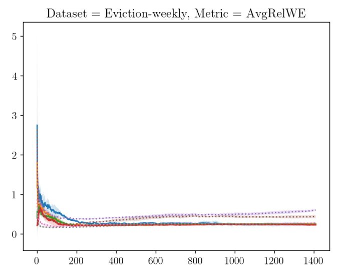
Let us first focus on the Adult dataset and privacy budget $\varepsilon \geq 1$, using the block counter in Algorithm 14 is typically better than using the simple counter. However, for $\varepsilon = 0.5$, we see that the simple counter performs better. The results of experiments over the Eviction dataset do not conclude if the simple counter is better than the block counter for any particular ε. We believe that the high variance of the block counter at the beginning of time, together with the selection error due to the Exponential mechanism, leads to such behavior.



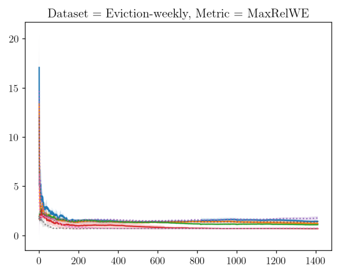
(a) Average of workload errors



(b) Maximum of workload errors

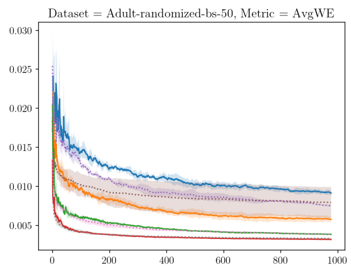


(c) Average of relative workload errors

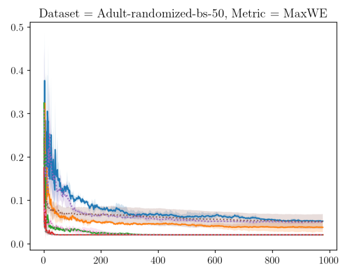


(d) Maximum of relative workload errors

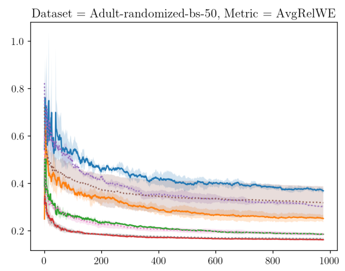
FIGURE 4.7. Simple vs block counter for the Eviction-weekly dataset



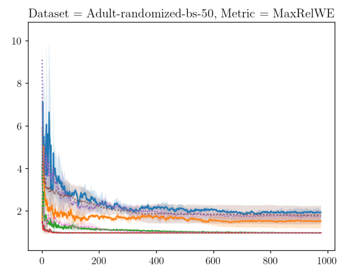
(a) Average of workload errors



(b) Maximum of workload errors

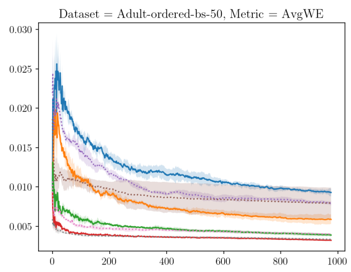


(c) Average of relative workload errors

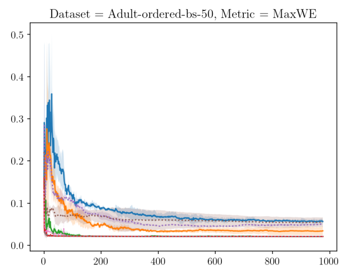


(d) Maximum of relative workload errors

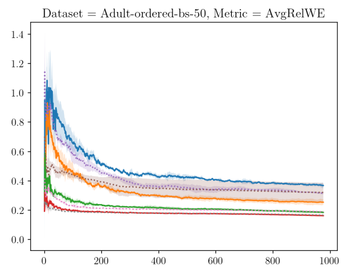
FIGURE 4.8. Simple vs block counter for the Adult-randomized-bs-50 dataset



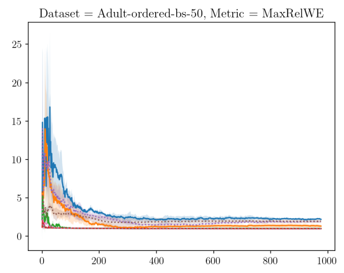
(a) Average of workload errors



(b) Maximum of workload errors



(c) Average of relative workload errors



(d) Maximum of relative workload errors

FIGURE 4.9. Simple vs block counter for the Adult-ordered-bs-50 dataset

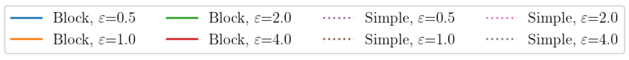


Table 4.4. Query error metrics comparing the simple and block counters in the proposed method for the Eviction dataset across various privacy budgets.

Dataset	ε	**Metric**	**Simple**	**Block**
Eviction-weekly	0.5	AvgRelWE	0.6024	**0.2545**
		AvgWE	0.0191	**0.0092**
		MaxRelWE	1.7817	**1.4485**
		MaxWE	**0.0467**	0.0507
	1.0	AvgRelWE	0.4409	**0.2553**
		AvgWE	0.014	**0.0136**
		MaxRelWE	1.3978	**1.2361**
		MaxWE	**0.0436**	0.057
	2.0	AvgRelWE	0.2746	**0.2376**
		AvgWE	**0.0079**	0.0126
		MaxRelWE	**0.7573**	1.121
		MaxWE	**0.0154**	0.044
	4.0	AvgRelWE	0.2632	**0.2284**
		AvgWE	**0.0075**	0.012
		MaxRelWE	0.71	**0.71**
		MaxWE	**0.0162**	0.0411

Table 4.5. Query error metrics comparing the simple and block counters in the proposed method for the Adult datasets across various privacy budgets.

Dataset	ε	**Metric**	**Simple**	**Block**
Adult-ordered-bs-50	0.5	AvgRelWE	**0.3165**	0.3698
		AvgWE	**0.008**	0.0093
		MaxRelWE	**1.9028**	2.2472
		MaxWE	**0.0497**	0.0572
	1.0	AvgRelWE	0.3214	**0.2552**
		AvgWE	0.008	**0.0059**
		MaxRelWE	1.9528	**1.3285**
		MaxWE	0.0547	**0.0338**
	2.0	AvgRelWE	0.1904	**0.1864**
		AvgWE	0.004	**0.0039**
		MaxRelWE	**0.9798**	1.0114
		MaxWE	0.0208	**0.0208**
	4.0	AvgRelWE	0.1667	**0.1632**
		AvgWE	0.0033	**0.0032**
		MaxRelWE	0.978	**0.978**
		MaxWE	0.0208	**0.0208**
Adult-randomized-bs-50	0.5	AvgRelWE	**0.3035**	0.3718
		AvgWE	**0.0075**	0.0092
		MaxRelWE	**1.7575**	1.9395
		MaxWE	**0.0504**	0.0525
	1.0	AvgRelWE	0.3191	**0.2534**
		AvgWE	0.0079	**0.0058**
		MaxRelWE	1.8162	**1.5294**
		MaxWE	0.0514	**0.0383**
	2.0	AvgRelWE	0.1863	**0.1862**
		AvgWE	0.0039	**0.0039**
		MaxRelWE	**0.977**	0.9804
		MaxWE	0.0208	**0.0208**
	4.0	AvgRelWE	0.1673	**0.1629**
		AvgWE	0.0033	**0.0032**
		MaxRelWE	**0.9778**	0.9779
		MaxWE	0.0208	**0.0208**

CHAPTER 5

Differentially Private Synthetic Trajectories

5.1. Introduction and motivation

In Chapters 3 and 4 we introduced the problem of generating synthetic streams. However, we made an assumption that the number of contributions a user can make throughout time is limited. Here, we consider datasets where the users may contribute many points throughout time resulting in trajectories. In this work, we only consider the task of one-time release.

Many existing works have considered the task of differentially private release of synthetic trajectories such as [33,34,37,64,70]. However, these methods have not been adopted for real-world datasets and a key challenge is that many such methods use a coarse discretization grid which does not capture the high-resolution correlations in the trajectories. In this work, we discuss some of the challenges in adopting a high-resolution grid and discuss our proposed solution to these challenges.

5.2. Problem setup

Similar to previous chapters, let $\mathcal{X}$ be a space of points. A trajectory of length t is a sequence of t points in $\mathcal{X}$, that is, it is an element of the set $\mathcal{X}^t$. We represent trajectories using a bold lowercase letter such as $\boldsymbol{x}$. If $\boldsymbol{x} \in \mathcal{X}^t$ we represent it as a tuple $(x_1, x_2, \dots, x_t)$. Let $\mathcal{X}^* \coloneqq \cup_{t \in \mathbb{N}} \mathcal{X}^t$ be the set of all possible trajectories. Let $f : \mathcal{X}^* \to \mathbb{N}$ be a private dataset that is a collection of trajectories, of possibly different lengths, such that $f(\boldsymbol{x}) \in \mathbb{N}$ represents the multiplicity of $\boldsymbol{x} \in \mathcal{X}^*$ in the dataset. We aim to generate an accurate synthetic dataset $g : \mathcal{X}^* \to \mathbb{N}$ that is representative of f, while still preserving the privacy of individuals who contributed to f.

In this case, we will simply use the classic definition of differential privacy as presented in Definition 2.1.2. Thus, by Definition 2.1.1 of neighboring datasets, a neighboring dataset $\tilde{f} : \mathcal{X}^* \to \mathbb{N}$ for f can be constructed by either increasing or decreasing the multiplicity of exactly one trajectory by 1. Note that this setup is challenging as, even though we restrict that neighboring datasets might differ by exactly one trajectory, but that one trajectory may contain every point of $\mathcal{X}$ and multiple times over. In other words, we do not restrict the length of the trajectories.

5.2.1. Some terminologies.

5.2.1.1. *Regular trajectory.* A trajectory $\boldsymbol{x} = (x_1, x_2, \dots, x_t)$ is regular if no point repeats in the trajectory, that is, $x_i \neq x_j$ for all $i, j \in [t]$ if $i \neq \jmath$.

5.2.1.2. *Self-loop.* A trajectory $\boldsymbol{x} = (x_1, x_2, \dots, x_t)$ is said to contain a self-loop at some location $i \in [t-1]$ if $x_i = x_{i+1}$, that is the two consecutive points at index i are the same.

5.3. Preliminaries and some related works

In this section, we will look at some of the existing approaches for differentially private synthetic trajectory generation. We will also briefly go over the mathematical tools used by these techniques. In Section 5.5, we will use these tools and build upon existing techniques when providing our method.

5.3.1. Markov model assumption. For simplicity, let us first assume that the space $\mathcal{X}$ is discrete and finite. To model the trajectory generation process, we assume that there is an underlying probability distribution that the true dataset follows. In particular, we assume that any trajectory, say $\boldsymbol{x} = (x_1, x_2, \dots, x_t)$, is generated as a discrete-time Markov chain using a sequence of random variables $X_1, X_2, \dots, X_t$ such that at any time $i \in [t]$ the probability $\mathbb{P}\{X_i = x_i\}$ is only a function of the observations $x_1, x_2, \dots, x_{i-1}$. Moreover, we assume that the underlying distribution follows a first-order Markov process such that

$$\mathbb{P}\left\{X_i = x_i \mid X_1 = x_1, X_2 = x_2, \dots, X_{i-1} = x_{i-1}\right\} = \mathbb{P}\left\{X_i = x_i \mid X_{i-1} = x_{i-1}\right\}, \tag{5.1}$$

for all $i \in [2, t]$. Furthermore, we assume that this transition probability does not depend on time and thus

$$\mathbb{P}\left\{X_i = x_i \mid X_{i-1} = x_{i-1}\right\} = A_{x_{i-1}, x_i}, \tag{5.2}$$

for all $i \in [2, t]$, where A is a transition matrix independent of time (trajectory length). Many existing works in the literature such as [33, 34, 37, 64, 70] follow these assumptions. [70] have also explored using a second-order Markov model in conjunction with a first-order model. In this work, however, we will be limiting to first-order only.

One may feel that an assumption of the Markov property is far from reality. Indeed, in many cases, trajectories depend not simply on the past locations but also on the future locations, and in particular the end location. For example, if we are driving our car from home to work, we likely have a predefined trajectory to follow based on the start location (home) and end location (work). Some existing approaches such as [37,70] only account for the end location indirectly, by assuming all trajectories start and end in some predefined virtual end-points and generating trajectories between these end-points (more details in Section 5.3.5). Other works such as [33,34,64] directly incorporate the end location. In these cases, the trajectory is still assumed to be generated as a Markov chain but conditioned on some end location. We will discuss this in more detail in Section 5.3.7.

A key task in all of these works is thus the generation of the transition matrix A while preserving differential privacy. Note that the dimensions of A are $|\mathcal{X}| \times |\mathcal{X}|$. However, the space $\mathcal{X}$ may be a high-dimensional space. In fact, in many applications, $\mathcal{X}$ is a continuous domain (such as the collection of latitude and longitude pairs). In both of these cases, it is difficult to model the Markov model in a meaningful way while guaranteeing differential privacy. A popular approach to deal with this problem is via a discretization of the space $\mathcal{X}$.

5.3.2. Discretization. In order to limit the number of possible transition states in the Markov process, we transform the trajectories on the (high-dimensional or continuous) space $\mathcal{X}$ to trajectories on a low-dimensional space discrete and finite space Σ. We describe this transformation below.

Let Σ be a finite partition of the space $\mathcal{X}$, that is $\cup_{\sigma\in\Sigma}\sigma = \mathcal{X}$ and $\sigma_i \cap \sigma_j = \emptyset$ for all $\sigma_i, \sigma_j \in \Sigma$ if $\sigma_i \neq \sigma_j$. Then we define the transformation of any trajectory $\boldsymbol{x} = (x_1, x_2, \ldots, x_t) \in \mathcal{X}^t$ using a mapping $\Phi : \mathcal{X}^* \to \Sigma^*$ such that $\Phi(\boldsymbol{x}) = (\sigma_1, \sigma_2, \ldots, \sigma_t) \in \Sigma^t$ and $\sigma_i \ni x_i$ for all $i \in [t]$. Hence, each point in a trajectory is replaced by the subset of $\mathcal{X}$ in Σ that contains the point. We represent the transformed trajectory dataset using $F : \Sigma^* \to \mathbb{N}$ such that

$$F(\boldsymbol{\sigma}) \coloneqq \sum_{\boldsymbol{x}\in\mathcal{X}^*} f(\boldsymbol{x}) \cdot \mathbb{1}_{(\Phi(\boldsymbol{x})=\boldsymbol{\sigma})}, \tag{5.3}$$

for all $\boldsymbol{\sigma} \in \Sigma^*$.

We have now reduced the problem of generating a trajectory over $\mathcal{X}$ to a problem of generating a trajectory over Σ. Let $G : \Sigma^* \to \mathbb{N}$ be the trajectory over Σ that is generated using F with differential privacy guarantees. After that, we can convert G to $g : \mathcal{X}^* \to \mathbb{N}$ using an inverse randomized transformation mapping similar in principle to the sampling method mentioned in

(3.2) in Chapter 3. Let $\Phi^{-1} : \Sigma \to \mathcal{X}$ be a randomized inverse transform such that $\Phi^{-1}(\sigma) = x$ for a random point x in σ. Then, with some abuse of notation, we can extend Φ^{-1} from points to trajectories as $\Phi^{-1} : \Sigma^t \to \mathcal{X}^t$ such that $\Phi^{-1}(\boldsymbol{\sigma}) = \left(\Phi^{-1}(\sigma_i)\right)_{i\in[t]}$. Finally we convert G to g such that $g(\boldsymbol{x})$ is the total number of trajectories $\boldsymbol{\sigma}$ in G such that a randomized inverse $\Phi^{-1}(\boldsymbol{\sigma})$ is $\boldsymbol{x}$.

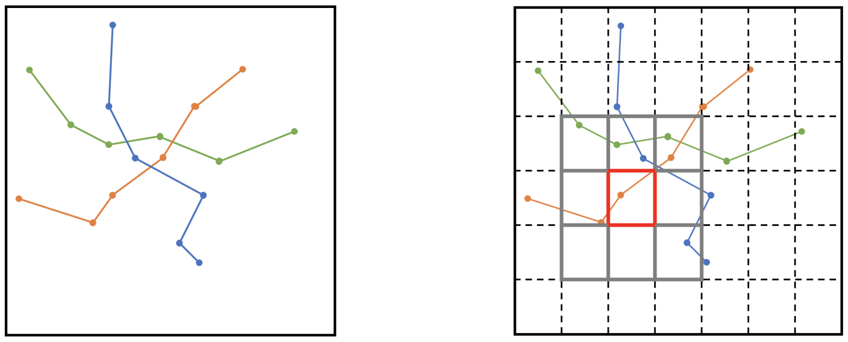

(a) Trajectories on a 2-D plane. (b) A uniform partition of the space.

FIGURE 5.1. An example uniform partition of the space. For the cell outlined with red, its neighbors are outlined grey.

Although these discussions are valid for more complex spaces as well, for simplicity, let us assume that $\mathcal{X}$ is a two-dimensional continuous rectangular space. Then, an example partition space Σ can be a rectangular grid over $\mathcal{X}$ such that all cells in the grid are congruent, see Figure 5.1. We will use the term *uniform grid* to emphasize that all cells are congruent.

We can model the trajectory generation via transitions among the $|\Sigma|$ states in Σ. Thus the complexity of the model will depend on the size $|\Sigma|$. In the case of a uniform grid, limiting the value of $|\Sigma|$ implies that individual cells have a large area. However, such discretization can become very coarse and lose important spatial correlation among points within the trajectory.

A natural way to keep $|\Sigma|$ small while still using fine partitions of the space $\mathcal{X}$ is to use density-aware and non-uniform partitions. Here, by non-uniform, we imply that the area of each subset within the partition may be different. Moreover, by density-aware, we imply that if the density of a region is higher, a finer partition in the region may be required so that discretization has a relatively lower loss of spatial information.

Many existing works indeed use density-aware partitions where the density of the private trajectory dataset is used. A popular approach is to use multi-level partitions and in particular two-level partitions. First, a coarse partition of the space is created using a uniform grid with large cells. Then, for each cell individually, the number of trajectory points in the cell is estimated using an approach similar to the Laplace Mechanism. Since trajectories can be of any finite length, the number of points a user contributes to a cell is not limited. In order to still be able to use the Laplace Mechanism, each point x_i in some trajectory $x \in \Sigma^t$ is only assumed to contribute a count of $1/t$ instead of 1 for a cell it belongs to so that the sensitivity of this construction is still bounded by 1.

Finally, this estimate is used to decide if, and the number of sub-cells into, the cell may be divided. This approach (or something similar) is seen in [33, 34, 70]. [37] on the other hand uses multiple uniform girds at different resolutions in a hierarchical system. The varying resolutions intuitively capture trajectories moving at different speeds. Note that since these processes of discretization depend on the true data, they must also satisfy differential privacy.

[64] circumvents spending privacy budget on discretization by creating partitions based on public information about the space $\mathcal{X}$. In particular, they consider road networks within a city, the first (coarse) level partition is based on the regional partition of the city (say into districts), and the finer level partition is based on the density of the underlying road network within each of the regions. While this method is helpful in discretization without spending any privacy budget, it assumes that the given trajectory dataset concurs with the public information which may not always be the case.

5.3.3. Additional notations. Before moving further we need to introduce some notations. For convenience, we will refer to the elements of Σ as *cells*. Also, for any $\boldsymbol{\sigma} \in \Sigma^*$ let $|\boldsymbol{\sigma}|$ denote the length of the trajectory $\boldsymbol{\sigma}$.

5.3.3.1. *Restriction of a trajectory.* For any trajectory $\boldsymbol{\sigma} \in \Sigma^*$ and ordered set of indices $I \subseteq [|\boldsymbol{\sigma}|]$, we define the restriction of $\boldsymbol{\sigma}$ on I as,

$$\boldsymbol{\sigma}|_I \coloneqq (\sigma_i)_{i \in I}\,.$$

5.3.3.2. *Sub-trajectory.* Let $\boldsymbol{\sigma} \in \Sigma^*$ be a trajectory. We say that another trajectory $\boldsymbol{\sigma'} \in \Sigma^t$ is a sub-trajectory of $\boldsymbol{\sigma}$ if $|\boldsymbol{\sigma'}| \leq |\boldsymbol{\sigma}|$ and there exists $i \in [|\boldsymbol{\sigma}| + 1 - t]$ such that $\boldsymbol{\sigma}|_{[i,i+t-1]} = \boldsymbol{\sigma'}$. That is if $\boldsymbol{\sigma'}$ is a contiguous subsequence of $\boldsymbol{\sigma}$. We will use the subset notation, that is $\boldsymbol{\sigma'} \subset \boldsymbol{\sigma}$, to say that $\boldsymbol{\sigma'}$ is a sub-trajectory of $\boldsymbol{\sigma}$.

Furthermore, let $C_{subset} : \Sigma^* \times \Sigma^* \to \mathbb{N}$ be an operator such that for any pair of $\boldsymbol{\sigma'}, \boldsymbol{\sigma} \in \Sigma^*$, $C_{subset}(\boldsymbol{\sigma'}, \boldsymbol{\sigma})$ denotes the number of occurrences of $\boldsymbol{\sigma'}$ in $\boldsymbol{\sigma}$, that is,

$$C_{subset}(\boldsymbol{\sigma'}, \boldsymbol{\sigma}) := \begin{cases} \sum_{i=1}^{|\boldsymbol{\sigma}|+1-|\boldsymbol{\sigma'}|} \left(\mathbb{1}_{\boldsymbol{\sigma'}=\boldsymbol{\sigma}|_{[i,i+|\boldsymbol{\sigma'}|-1]}}\right), & \boldsymbol{\sigma'} \subset \boldsymbol{\sigma}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

5.3.3.3. *Number of occurrences as sub-trajectory in a dataset.* We define $C_{subtrajectories} : \Sigma^* \times \mathbb{N}^{\Sigma^*} \to \mathbb{N}$ to be an operator such that for any trajectory $\boldsymbol{\sigma} \in \Sigma^*$ and dataset $F : \Sigma^* \to \mathbb{N}$, $C_{subtrajectories}(\boldsymbol{\sigma}, F)$ denotes the total occurrences of $\boldsymbol{\sigma}$ as a sub-trajectory for trajectories in F, that is,

$$C_{subtrajectories}(\boldsymbol{\sigma}, F) = \sum_{\boldsymbol{\sigma'} \in \Sigma^*} F(\boldsymbol{\sigma}) \cdot C_{subset}(\boldsymbol{\sigma}, \boldsymbol{\sigma'}). \tag{5.5}$$

5.3.4. The transition probability. Once we have transformed f, the trajectory dataset of points, to F, the trajectory dataset of cells. We are now ready to talk about how to learn the probability of transition under the assumption of the Markov model with differential privacy. We first assume that, during trajectory generation, any transition from a cell $\sigma \in \Sigma$ is limited only to the cells adjacent to σ. We define adjacent cells in Definition 5.3.1

DEFINITION 5.3.1 (Adjacent cells). *Two cells $\sigma, \sigma' \in \Sigma$ are called adjacent if they share a boundary. That is, there exists some $x \in \mathcal{X}$ such that for all $\delta > 0$, $B(x, \delta) \cap \sigma \neq \emptyset$ and $B(x, \delta) \cap \sigma' \neq \emptyset$. Here, $B(x, r)$ is an open ball of radius r centered at x.*

Furthermore, for any cell $\sigma \in \Sigma$, we define the set of all cells adjacent to it as $N(\sigma)$. Figure 5.1 shows an example of all adjacent cells of a given cell for a uniform grid. Note that in this case, there are only 9 possible cells (including the highlighted cell itself) where the transition could happen. With this assumption, we have drastically reduced the complexity of our model as the number of possible cells where the next transition in trajectory can happen changed from $|\Sigma|$ to $\mathcal{O}(1)$. This constrained approach is used by many existing works including [33, 34, 37, 64, 70].

5.3.4.1. *Adjacency graph.* We can further represent this adjacency restriction using a weighted directed graph $\mathcal{G} = (\Sigma, E, w)$ such that for any pair of cells $\sigma_i, \sigma_j \in \Sigma$ there is an edge $(\sigma_i, \sigma_j) \in E$ if and only if σ_j is adjacent to σ_i. We also assume that self-loops are possible and thus $(\sigma, \sigma) \in E$ for all $\sigma \in \Sigma$. The set $N(\sigma)$ thus represents all the out-neighbors of σ as per $\mathcal{G}$.

In this representation, $w : E \to [0,1]$ is a mapping such that for any $e = (\sigma, \sigma') \in E$, $w(e)$ represents the probability of transition from cell σ to σ'. Thus it follows that for any $\sigma \in \Sigma$,

$$\sum_{\sigma' \in N(\sigma)} w(\sigma') = 1.$$

An empirical estimate of the transition probabilities can be constructed using the discretized dataset $F : \Sigma^* \to \mathbb{N}$ as,

$$w(\sigma, \sigma') \coloneqq \frac{C_{subtrajectories}\big((\sigma, \sigma'), F\big)}{\sum_{\sigma'' \in N(\sigma)} C_{subtrajectories}\big((\sigma, \sigma''), F\big)} \tag{5.6}$$

The task remains to find a differentially private estimate of this weight function w. Mostly all of the existing works mentioned previously (such as [33,34,37,64,70]) first estimate $C_{subtrajectories}\big((\sigma, \sigma'), F\big)$ for a given σ and all of its neighbors $\sigma' \in N(\sigma)$ using the Laplace mechanism. Similar to discretization, the number of transition counts a trajectory affects is dependent on the length of the trajectory. Hence, for a trajectory of length t, each transition is assumed to contribute only $1/(t-1)$ which in turn ensures that the overall sensitivity for the Laplace Mechanism is 1. We discuss this in more detail in Section 5.5.3

Then, the estimates are normalized by their sum (as in Equation (5.6)) to generate the probabilities. Let us denote this estimated weight map for the edges in $\mathcal{G}$ using $\hat{w}$. In this work, we also do something similar and we discuss the details more broadly in Section 5.5.3.

5.3.5. Sampling trajectories.

5.3.5. Sampling trajectories. Finally, we are at the stage where we have an estimate of the transition probabilities (in $\hat{w}$) and we want to sample synthetic trajectories. While the transition probabilities are estimated, we still need a cell where the trajectories start. Two key approaches have been explored to find the start cell and we discuss them in below subsections.

5.3.5.1. *(1) Using virtual start and end cells.* Methods such as [37, 70] create two virtual cells σ_{start} and σ_{end} and assume that all trajectories start and end at these cells respectively. Thus, the discretized space Σ is extended to a space (say Σ_+) such that $\Sigma_+ = \Sigma \cup \{\sigma_{start}, \sigma_{end}\}$. Furthermore, $\mathcal{G} = (\Sigma, E, w)$ is extended to $\mathcal{G}_+ = (\Sigma_+, E_+, w_+)$, Here, $E_+ = E \cup \big\{(\sigma_{start}, \sigma) \mid \sigma \in \Sigma\big\} \cup \big\{(\sigma, \sigma_{end}) \mid \sigma \in \Sigma\big\}$. In other words, it is possible to transition from σ_{start} to any other state in Σ. It is also possible to transition from any state in Σ to σ_{end}. w_+ can be recalculated using Equation (5.6), except accounting for all cells in Σ_+.

A differentially private estimate $\hat{w_+}$ can be constructed for w_+, as discussed previously. Furthermore, for sampling, we can simply start with $X_1 = \sigma_{start}$ and iteratively sample X_i as a Markov chain such that the probability that

$$\mathbb{P}\left\{X_i = \sigma_i \mid X_{i=1} = \sigma_{i-1}\right\} = \hat{w_+}\left(s_{i-1}, s_i\right),$$

for any $s_{i-1}, s_i \in \Sigma$. The chain terminates at length t if $X_t = \sigma_{end}$ or we meet a certain pre-defined maximum length.

There are two key problems with this approach: (1) $\hat{w_+}(\sigma_{start}, \cdot)$ is not a good estimate of the starting cells for the trajectories; and (2) there is no guarantee that the Markov chain random walk would converge to the end start σ_{end} and is likely a very random walk contrary to how trajectories are generated in real-life.

Let us discuss in more detail the problem (1) here. Note that the empirical probability $w_+(\sigma_{start}, \cdot)$ is a good approximation of the distribution of the true starting cells of the trajectories. However the estimate $\hat{w_+}(\sigma_{start}, \cdot)$ is likely biased. This is because to preserve privacy and use the Laplace Mechanism the contribution of any trajectory say $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_t) \in \Sigma^t$ to $\hat{w_+}(\sigma_{start}, \sigma_1)$ is assumed to be $1/t$. Thus the distribution becomes biased towards cells from where shorter length trajectories originate. [70] attempts to solve this problem by doing a bias correction using the graphical distance between any pair of states in the graph $\mathcal{G}_+$. However, the second problem mentioned above remains and we discuss how to handle that in the next subsection.

5.3.5.2. *Estimate start and end cell distribution, followed by conditional random walk.* In Section 5.3.1, we mentioned that many real-world trajectories are not generated simply based on the past locations but in particular the end location should play a key role. To accommodate the end location, a simple approach is to use the following three-step process: (1) estimate a distribution for the start and end cells of the private trajectories while preserving differential privacy; (2) sample start and end pairs based on this estimated distribution; and (3) for each start and end cell pair, sample trajectories as Markov chain starting from the start state and conditioned on reaching the end state. This high-level process is used in works such as [33, 34, 64]. We also use this process in this work and hence we discuss it in detail in the next subsections.

5.3.6. Estimate end and start pair distribution. Let $C_{se} : \Sigma \times \Sigma \to \mathbb{N}$ be such that $C_{se}(\sigma, \sigma')$ denote the number of trajectories in the private data F that start at cell $\sigma \in \Sigma$ and end at cell $\sigma' \in \Sigma$. We can find this map for the true dataset as follows,

$$C_{se}(\sigma, \sigma') = \sum_{\boldsymbol{\sigma} \in \Sigma^*} F(\boldsymbol{\sigma}) \cdot \mathbb{1}_{(\sigma_1 = \sigma)} \cdot \mathbb{1}_{\left(\sigma_{|\boldsymbol{\sigma}|} = \sigma'\right)}. \tag{5.7}$$

To estimate this mapping while preserving privacy, many previous works such as [33, 64] simply use the Laplace Mechanism over the histogram of all possible pairs of cells. Let $\hat{C}_{se}$ denote the estimated mapping then by Laplace Mechanism we have,

$$\hat{C}_{se}(\sigma, \sigma') = C_{se}(\sigma, \sigma') + \text{Lap}\left(\frac{1}{\varepsilon}\right), \tag{5.8}$$

for all pairs of $\sigma, \sigma' \in \Sigma$.
Now, we can sample $\left\lceil \max\left\{\hat{C}_{se}(\sigma, \sigma'), 0\right\}\right\rceil$ trajectories for each pair of $\sigma, \sigma' \in \Sigma$. We discuss how to sample the trajectories in the next subsection.

5.3.7. Conditional random walk. Let us assume we want to generate a trajectory of length t as a Markov chain using the sequence of random variables $X_1, X_2, \ldots, X_t$ with the constraint that $X_1 = \sigma_1$ and $X_t = \sigma_t$ for some fixed $\sigma_1, \sigma_t \in \Sigma$. We will use the Markov property and the adjacency graph $\hat{\mathcal{G}} = (\Sigma, E, \hat{w})$ to generate this trajectory.
Let us assume that we have reached step i for some $1 < i < t$ following the conditional random walk. Moreover, we assume that we have observed the following trajectory so far $(\sigma_1, \sigma_2, \ldots, \sigma_{i-1})$, for some $\sigma_2, \ldots, \sigma_{i-1} \in \Sigma$. Then, for any $\sigma \in N(\sigma_{i-1})$ we have that,

$$\begin{aligned}
&\mathbb{P}\left\{X_i = \sigma \mid X_1 = \sigma_1, \ldots, X_{i-1} = \sigma_{i-1}, X_t = \sigma_t\right\} \\
&= \mathbb{P}\left\{X_i = \sigma \mid X_{i-1} = \sigma_{i-1}, X_t = \sigma_t\right\} = \mathbb{P}\left\{X_2 = \sigma \mid X_1 = \sigma_{i-1}, X_{t-i+2)} = \sigma_t\right\} \\
&= \frac{\mathbb{P}\left\{X_2 = \sigma \mid X_1 = \sigma_{i-1}\right\} \cdot \mathbb{P}\left\{X_{t-i+2} = \sigma_t \mid X_2 = \sigma\right\}}{\mathbb{P}\left\{X_{t-i+2} = \sigma_t \mid X_1 = \sigma_{i-1}\right\}} \\
&= \frac{\hat{w}\left(\sigma_{i-1}, \sigma\right) \cdot \hat{w}^{t-i}\left(\sigma, \sigma_t\right)}{\hat{w}^{t-i+1}\left(\sigma_{i-1}, \sigma_t\right)},
\end{aligned}$$

where for any $\sigma, \sigma' \in \Sigma$ and $k \in \mathbb{N}$, $\hat{w}^k(\sigma, \sigma')$ is the transition probability from state σ to σ' for the k step Markov chain such that $X_1 = \sigma$ and $X_{k+1} = \sigma'$.
Note that here we have assumed that even before generating the trajectory we knew the length t of the trajectory. Indeed to generate a conditional random walk as discussed above, we need to

know the length of the trajectory in advance. This leads to the final piece of the puzzle, which is estimating the length of the trajectory, and we discuss how to do that in the next subsection.

5.3.8. Estimate the length of trajectory for a start and end cell pair. [33] models the length of the trajectory given a start and end cells as an exponential distribution with scale $\ln 2/m(\sigma, \sigma')$, where, for any $\sigma, \sigma' \in \Sigma$, $m(\sigma, \sigma')$ is the median length of trajectories starting at σ and ending at σ' in the private dataset F. [33] estimates the median length m from the true data using the Exponential Mechanism to preserve differential privacy. [34] uses a similar technique but employs various distributions (including the exponential distribution) and picks the one that is closest to the true distribution based on some statistics.

[64] on the other hand does not spend any privacy budget on the sampling of trajectory lengths and instead samples a length l with probability proportional to $\hat{w}^{(l-1)}\left(\sigma, \sigma'\right)$, that is the estimated probability of having a Markov chain of length l between σ and σ'.

5.4. Limitations and our contribution

While there is an abundance of research related to differentially private synthetic trajectory generation, a key issue remains unsolved, which is how to generate realistic trajectories. A key challenge is that most existing methods only work on a very coarse discretization of the space $\mathcal{X}$. While some of the previously discussed methods such as [37] have supporting experiments for a fine discretization of the space $\mathcal{X}$, their sampling method is a random walk that does not produce a Markov chain conditional on the end location.

Other methods such as [33, 64] that use the conditional random walk are only compatible with a very coarse discretization of the space $\mathcal{X}$ and thus potentially lose a lot of spatial information in this step. For example, typically the order of $|\Sigma|$ for the first level grid as proposed in [33] is 10^2. For comparison, $|\Sigma|$ in the finest resolution used in [70] is of the order of 10^5. Indeed many challenges arise when using existing methods with conditional Markov chain at very fine discretization. We discuss these challenges in detail below.

First, let us focus on the challenges in estimating the start and end cell distribution, which is required to sample trajectories conditioned on the end state. As discussed in Section 5.3.6 this results in the estimation of a histogram $\hat{C}_{se} : \Sigma \times \Sigma \to \mathbb{N}$ via the Laplace Mechanism. There are two problems with this approach: (1) the memory required to store such a mapping can be very large if Σ is large, and (2) the true histogram C_{se} is likely a sparse histogram, but the addition of noise to all possible cell pairs results in a large number of non-zero entries.

Secondly, to generate a t length trajectory, the conditional Markov chain construction requires us to calculate the i-step probabilities $\hat{w}^i$ for all $i < t$. While the initial mapping $\hat{w}$ is restricted to neighboring cells, as we increase the number of steps, more and more cells become reachable from one another increasing the complexity of finding these transition probabilities. In other words, if $\hat{A}$ represents our estimated transition probability matrix, while the $\hat{A}$ is typically sparse, $\hat{A}^n$ quickly becomes a dense matrix of size $|\Sigma| \times |\Sigma|$ consuming a lot of memory and compute.

Another problem with many existing works is that they have not been explored over spaces with constraints. For example, the space $\mathcal{X}$ of points is almost always assumed to be a rectangular space where every point within $\mathcal{X}$ is possible to appear in the trajectories. However many spatial constraints exist in real-world datasets. For example, if a dataset contains trajectories obtained by cars driving in the city then the points in the trajectories are likely to be only at geographic locations where the road network of the city exists. This problem was also highlighted in [64].

5.4.1. Contributions. We follow a process very similar to what is discussed in Section 5.3 but we provide alternatives in almost all aspects of the process. These enhancements are aimed at overcoming the above-mentioned limitations. We summarize our contributions below.

- Our method works for many general spaces $\mathcal{X}$ as long as it can be partitioned hierarchically. In particular, $\mathcal{X}$ can be a graph representing the road network of a city. Thus we can accommodate most spatial constraints about $\mathcal{X}$.
- We use a density-based discretization approach which can handle datasets that are sparse or skewed. The approach produces partitions of very high resolution in dense areas while maintaining a relatively smaller number of total cells in Σ.
- We also provide an approach on how to estimate the sparse histogram C_{se} while preserving differential privacy and high utility.
- Furthermore, we enhance the conditioned Markov chain sampling approach and circumvent the generation of large powers of the estimated transition matrix $\hat{A}$.
- We also give a new fast sampling method that can be used to generate synthetic trajectories conditioned on end-state.

Algorithm 16 Discretization using PrivTree_T

```
1: Input: $f : \mathcal{X}^* \to \mathbb{N}$, a hierarchical decomposition $T$ of $\mathcal{X}$, the privacy budget parameter $\varepsilon_1$,
   the threshold count for a node $\theta$.
2: Output: $\Sigma$, a partition of $\mathcal{X}$.
3: Extend $f$ to $H : V(T) \to \mathbb{N}$ via Equation (5.9).
4: Set $S \leftarrow \text{PrivTree}_T(H, \varepsilon_1, \theta)$.
5: Set $\Sigma \leftarrow \mathcal{L}(S)$.
```

- We present a variety of experiments over real-world datasets that validate our proposed method.

5.5. Proposed Method

In this section, we discuss our method which follows a pipeline very similar to existing works as discussed in Section 5.3. Our entire method can be summarized in the following steps:

(1) a differentially private density-aware discretization of the space $\mathcal{X}$ into Σ;

(2) construction of an adjacency graph $\mathcal{G} = (\Sigma, E, w)$;

(3) while preserving differential privacy, find $\hat{w}(\sigma, \sigma')$, an estimate of the transition probability $w(\sigma, \sigma')$, for any pair of $\sigma \in \Sigma$ and $\sigma' \in N(\sigma)$;

(4) while preserving differential privacy, find $\hat{C}_{se}(\sigma, \sigma')$, an estimate of the number of trajectories starting and ending at σ and σ' respectively, for any pair of $\sigma, \sigma' \in \Sigma$;

(5) sample trajectories in the space Σ based on $\hat{C}_{se}$ and $\hat{w}$, creating the dataset $G : \Sigma^* \to \mathbb{N}$;

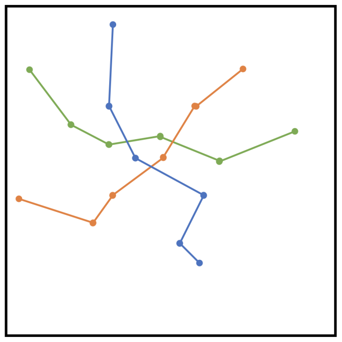
(6) convert G to the $g : \mathcal{X}^* \to \mathbb{N}$, a dataset of trajectories in $\mathcal{X}$.

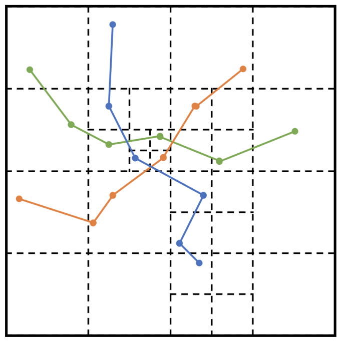
Note that only the steps (1), (3), and (4) mentioned above consume the privacy budget. Thus we will split our total privacy budget, say ε, into three parts ε_1, ε_2, and ε_3 to use in these three steps respectively. We discuss how to choose this split later. In the subsections below, we discuss all steps in detail.

5.5.1. Discretization using PrivTree.

We use PrivTree [79] to discretize the space $\mathcal{X}$ into Σ. As discussed in Chapter 3, PrivTree is a remarkable algorithm that can create a density-aware space partition. Moreover, PrivTree can be used for most spaces $\mathcal{X}$, as long as there exists a hierarchical decomposition of the space.

We show an example hierarchical partition in Figure 5.2. Figure 5.2a shows three trajectories with different colors on a 2-dimensional space. Note that the trajectories are represented as points but


(a) Trajectories on a 2-D plane.


(b) A density-based partition of the space.

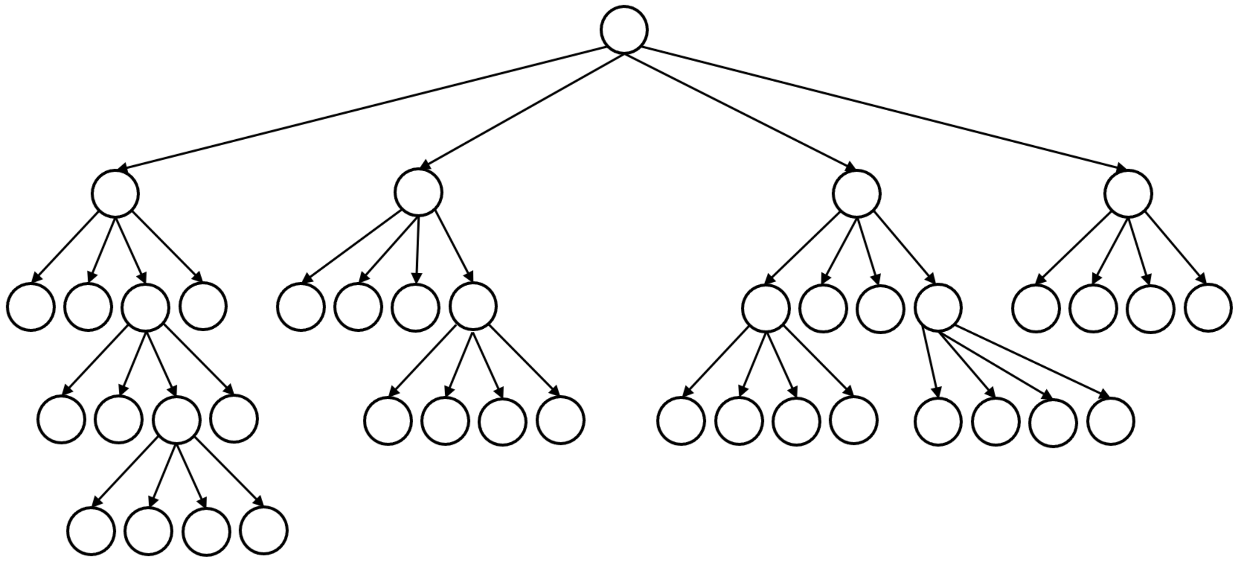

(c) Hierarchical decomposition corresponding to the suggested partition.

FIGURE 5.2. An example partition using density-based hierarchical decomposition of the space such that each cell has at most one point.

are connected by a straight line. Figure 5.2b shows a partition of the space such that each cell has at most one point from any trajectory. Figure 5.2c shows a hierarchical partition that results in the partition in Figure 5.2b. Let the root node of the hierarchical partition represent the entire space. Each node has exactly 4 children that partition the space defined by the parent node into 4 equal subspaces. The children are ordered clockwise starting from the top-left child.

We utilize Algorithm 5 from Chapter 3 for discretization. Recall that the algorithm is instantiated for a tree T which represents a hierarchical decomposition of the space $\mathcal{X}$ and produces a subtree S of T. The density aware partition is done based on the input dataset (say) $H : V(T) \to \mathbb{N}$ where $H(v)$ denotes the number of points in the subregion $v \subseteq \mathcal{X}$. Once we have the subtree S, we define

$\Sigma = \mathcal{L}(S)$, which is the set of all leaves in S. We present this process in Algorithm 16 and discuss the relevant details below.

To use the PrivTree algorithm we need to have two things: (1) the largest possible tree T, and (2) a dataset (say) $H : V(T) \to \mathbb{N}$. We will skip the discussion on how T is created for now and come back to it when we discuss the experiments in Section 5.6. Let us now focus on creating the dataset H given the trajectory dataset $f : \mathcal{X}^* \to \mathbb{N}$. This can be calculated as,

$$H(v) := \sum_{t=1}^{\infty} \sum_{\boldsymbol{x} \in \mathcal{X}^t} \left(f(\boldsymbol{x}) \cdot \left(\sum_{i=1}^{t} \mathbb{1}_{(x_i \in v)} \right) \right),$$

for all nodes $v \in V(T)$.

However, each trajectory may contribute multiple points in any node of the tree T. Thus the construction results in an unbounded sensitivity. To see this, let f and $\tilde{f}$ be a pair of neighboring datasets that differ in the count of some trajectory $\boldsymbol{x} \in \mathcal{X}^t$. Let H and $\tilde{H}$ be their respective extensions on the nodes of $V(T)$. Then, indeed we have that,

$$\left\| H - \tilde{H} \right\|_{C(T)} = \sum_{v \in \mathcal{L}(T)} \left| H(v) - \tilde{H}(v) \right| = \sum_{v \in \mathcal{L}(T)} \left(\sum_{i=1}^{t} \mathbb{1}_{(x_i \in v)} \right) = t,$$

which grows with the length of the trajectory. Here $\|\cdot\|_{C(T)}$ is the norm defined in Equation (3.4) over the space of consistent trees $C(T)$.

To limit the sensitivity of the algorithm we instead define the extension H as,

$$H(v) := \sum_{t=1}^{\infty} \sum_{\boldsymbol{x} \in \mathcal{X}^t} \left(f(\boldsymbol{x}) \cdot \left(\sum_{i=1}^{t} \frac{\mathbb{1}_{(x_i \in v)}}{t} \right) \right), \tag{5.9}$$

for all nodes $v \in V(T)$. Note the division by the length t of a trajectory $\boldsymbol{x} \in \mathcal{X}^t$. With this definition, for the example neighboring datasets above, we have that

$$\left\| H - \tilde{H} \right\|_{C(T)} = \sum_{v \in \mathcal{L}(T)} \left| H(v) - \tilde{H}(v) \right| = \sum_{v \in \mathcal{L}(T)} \left(\frac{1}{t} \sum_{i=1}^{t} \mathbb{1}_{(x_i \in v)} \right) = 1.$$

This leads to the following guarantees about the privacy of the discretization process.

Theorem 5.5.1 (Privacy of discretization). *Algorithm 16 satisfies ε_1-differential privacy.*

Algorithm 17 Calculate transition probabilities

```
1: Input: $F : \Sigma^* \to \mathbb{N}$, the adjacency graph $\mathcal{G} = (\Sigma, E, w)$, the privacy budget parameter $\varepsilon_2$, and
   $\alpha : \mathbb{N} \times \mathbb{N} \to [0,1]$, a map to determine the weight of a transition in a trajectory.
2: Output: $\hat{w} : E \to [0,1]$, an estimate of transition probabilities $w$.
3: // Normalized count for occurrences of edges in $E$
4: Initialize $c(e) = 0$ for all $e \in E$.
5: for $\boldsymbol{\sigma} \in \Sigma^*$ with $F(\boldsymbol{\sigma}) > 0$ do
6:     for $i = 1, 2, \ldots, |\boldsymbol{\sigma}| - 1$ do
7:         Set $c(\sigma_i, \sigma_{i+1}) \leftarrow c(\sigma_i, \sigma_{i+1}) + \alpha\left(|\boldsymbol{\sigma}| - 1, i\right) \cdot F(\boldsymbol{\sigma})$
8: // Find an estimated count
9: Initialize $\eta_e \sim \text{Lap}\left(\frac{1}{\varepsilon_2}\right)$ independently for all $e \in E$.
10: Set $\hat{c}(e) \leftarrow \eta_e$ for all $e \in E$.
11: // Non-negativity
12: Set $\hat{c}(e) \leftarrow \max\left\{c(e), 0\right\}$ for all $e \in E$.
13: // Normalization
14: for $\sigma \in \Sigma$ do
15:     Set $m_\sigma \leftarrow \sum_{\sigma' \in N(\sigma)} \hat{w}\left(\sigma, \sigma'\right)$
16:     for $\sigma' \in N(s)$ do
17:         Set $\hat{w}\left(\sigma, \sigma'\right) \leftarrow \left(\frac{1}{m_\sigma}\right) \cdot \hat{c}\left(\sigma, \sigma'\right)$
```

PROOF. Let f and $\tilde{f}$ be any pair of neighboring datasets as per Definition 2.1.1. Let H and $\tilde{H}$ be their corresponding extensions to $V(T)$ respectively as per Equation (5.9). Then, by construction, $\left\|H - \tilde{H}\right\|_{C(T)} = 1$. Hence, by Theorem 3.3.1 we have that this application of PrivTree$_T$ (Algorithm 5) to obtain S satisfies ε_1-differential privacy. The creation of Σ is simply a post-processing step and does not violate the privacy gurantees. □

5.5.2. Trajectory over Σ and graph $\mathcal{G}$. Once we have Σ, a partition of the space $\mathcal{X}$ we extend the input dataset $f : \mathcal{X}^* \to \mathbb{N}$ to $F : \Sigma^* \to \mathbb{N}$ using Equation (5.3). Note that if f and $\tilde{f}$ are neighbors as per Definition 2.1.1, their extensions, (say) F and $\tilde{F}$ respectively, are also neighbors. Next, we create the adjacency graph $\mathcal{G} = (\Sigma, E, w)$. E contains pair of cells in Σ that are adjacent as per Definition 5.3.1. We will discuss how to check for adjacent cells in practice in Section 5.6 which describes the experiments. The mapping $w : E \to [0,1]$ for the weights of the edges is calculated as per Equation 5.6.

5.5.3. Estimating the transition probabilities. We provide our algorithm for generating the transition probabilities in Algorithm 17. We follow an approach very similar to the existing methods and discussed in Section 5.3.4. In the algorithm, $c : E \to \mathbb{N}$ is a function such that $c(e)$ denotes the *normalized* count of any edge e in E. We will use the Laplace Mechanism to find an estimate $\hat{c}$ of c. Since a trajectory is made up of multiple edges (transitions), we limit the contribution of a trajectory for a particular edge by counting only a fraction toward each edge in the trajectory. This fraction is what we refer to as the normalized count.

Let $\alpha : \mathbb{N} \times \mathbb{N} \to [0, 1]$ be a function such that $\alpha(t, i)$ is the weight associated with the ith edge in a trajectory with t edges, such that for any $t \in \mathbb{N}$, $\sum_{i=1}^{t} \alpha(t, i) = 1$. Thus for any trajectory $\boldsymbol{\sigma} = (s_1, s_2, \ldots, s_t) \in \Sigma^t$, for any $i \in [1, t-1]$, the contribution to $c(\sigma_i, \sigma_{i+1})$ is given as $\alpha(t-1, i) \cdot F(\boldsymbol{\sigma})$. We refer to α as *sensitivity distribution*.

Previous methods have used a constant value for all edges in a trajectory such that $\alpha(t, i) = 1/t$ for all $t \in \mathbb{N}$ and $i \in [t]$. In this work, we also explore using a non-uniform weight for transitions in a trajectory. We explain the reasoning behind non-uniform weights with an example. Suppose we have trajectories in a space where there is a central region of high-density (say a downtown) area and there are some low-density neighborhoods (say residential areas) very far from the center where only a few of the trajectories originate or end and most of them go through the central region. Then, while many trajectories capture the transition pattern in the center, the transition patterns close to these neighborhoods may not be captured so well because of a low number of trajectories in those regions. Moreover, since these trajectories will typically be of large length, their contributions to edge counts are further penalized more so than for the trajectories in the denser regions with short-length trajectories. Hence, we explore non-uniform weights where the start and end transitions have more weight than others.

We suggest using an *Exponential decay* such that the weight is proportional to a quantity that decreases exponentially as we move away from the two ends of a trajectory. More formally, for any $t \in \mathbb{N}$ and $i \in [t]$ we set

$$\alpha(t, i) \propto \frac{1}{2^{\left|\lfloor t/2 \rfloor - i\right|}}. \tag{5.10}$$

With these explanations, we are now ready to talk about the privacy of Algorithm 17.

THEOREM 5.5.2 (Privacy of estimating transition probabilities). *Algorithm 17 satisfies ε_2-differential privacy.*

PROOF. Let f and $\tilde{f}$ be any pair of neighboring datasets as per Definition 2.1.1. Let us assume that they differ in the count of some trajectory $\boldsymbol{\sigma} \in \Sigma^t$. Let c and $\tilde{c}$ be their corresponding normalized edge count maps in Algorithm 17. Then we have,

$$\|c-\tilde{c}\|_{\ell_1} = \sum_{i=1}^{t-1} \alpha(t-1, i) = 1$$

Hence, by the privacy guarantees of the Laplace Mechanism (Theorem 2.2.4) the construction of $\hat{c}$ satisfies ε_2-differntial privacy. The creation of $\hat{w}$ is simply a post-processing step over $\hat{c}$ and does not violate the privacy guarantees. □

5.5.4. Estimating trajectory start and end state pairs. As discussed in Seciton 5.3.6, we want to estimate the number of trajectories that start and end for each pair of cells in Σ. If each possible pair of states is a *bin*, this results in a histogram query. However, simply using the Laplace mechanism to estimate the counts for this histogram query will result in poor utility. This is because the histogram will likely be sparse since we have a high-resolution discretization, which results in a large number of possible state pairs. The problem is that adding noise to the count of all possible pairs in this sparse histogram will result in multiple non-zero entries. The question thus becomes, how do we estimate a sparse histogram with differential privacy?
The idea is to convert the histogram from a sparse to a dense histogram by grouping some of the bins such that each group will likely have a non-zero value. Note that the algorithm that would create this grouping must make use of the spatial information about the states. Thus we rely again on *PrivTree* to create this spatial partition of the space $\Sigma \times \Sigma$. Algorithm 19 provides our algorithm which uses PrivTree and Algorithm 18. We have used Algorithm 18 as a subroutine to construct the inputs required for PrivTree. We discuss both of these algorithms in the below subsections.

5.5.4.1. *Constructing* T_{Σ^2}. The first thing we need for PrivTree is the largest possible tree (the variable T in Algorithm 5). Note that each leaf of this tree should be a value in $\Sigma \times \Sigma$. So, we name this tree T_{Σ^2}. We define its construction based on the output S of Discretization (Algorithm 16) so that the spatial information is used in our construction.
Algorithm 18 provides our method formally. We start the construction by setting the root node of T_{Σ^2} as $(root(S), root(S))$. Then, we recursively grow the tree such that for any node (u, v) already created in T_{Σ^2}, we set its children as the Cartesian product of $children(u, S)$ and $children(v, S)$. Here, we have used the notation $chldren(\cdot, S)$ and $children(\cdot, \cdot, T_{\Sigma^2})$ to emphasize that the children

Algorithm 18 Construction of T_{Σ^2}

```
1: Input: The discretized space $\Sigma$, the dataset of trajectories $F : \Sigma^* \to \mathbb{N}$, a hierarchical partition $S$ of $\mathcal{X}$.
2: Output: $T_{\Sigma^2}$, a hierarchical partition of $\mathcal{X} \times \mathcal{X}$, and $J : V(T_{\Sigma^2}) \to \mathbb{N}$ representing the count of points in each node in $T_{\Sigma^2}$.

3: // Constructing the tree
4: Create an empty tree $T_{\Sigma^2}$.
5: Set $root(T_{\Sigma^2}) \leftarrow (root(S), root(S))$.
6: for a node $(u, v) \in T_{\Sigma^2}$, recursively, do
7:     if $children(u) \neq \emptyset$ or $children(v) \neq \emptyset$ then
8:         if $children(u, S) \neq \emptyset$ then
9:             Set $B_1 \leftarrow children(u, S)$
10:        else
11:            Set $B_1 \leftarrow \{u\}$.
12:        if $children(v, S) \neq \emptyset$ then
13:            Set $B_2 \leftarrow children(v, S)$
14:        else
15:            Set $B_2 \leftarrow \{v\}$.
16:        Set $children(u, v, T_{\Sigma_2}) \leftarrow B_1 \times B_2$.

17: // Constructing the count map
18: Initialize the map $J \leftarrow 0$.
19: Set $J(u, v) \leftarrow C_{se}(u, v)$,calculated as per Equation (5.5), for all leaf node $(u, v) \in \mathcal{L}(T_{\Sigma^2})$
20: Extend the count to all internal nodes of $T_{\Sigma^2}$ as per the consistency requirement in Equation 3.3.
```

are as per the tree S and T_{Σ^2} respectively. The recursive process continues if any of u or v have children in S. If u or v does not have children, we use the set $\{u\}$ or $\{v\}$ respectively in the Cartesian product. Note the following about this construction,

- the set of nodes of T_{Σ^2} is $V(T_{\Sigma^2}) \subseteq V(S) \times V(S)$,
- the set of all leaves of T_{Σ^2} is $\mathcal{L}(T_{\Sigma^2}) = \mathcal{L}(S) \times \mathcal{L}(S) = \Sigma \times \Sigma$,
- if the fanout (maximum number of children of any node) of S is β, then the fanout of T_{Σ^2} is β^2

Once the tree T_{Σ^2} is created, we need to find a mapping $J : V(T_{\Sigma^2}) \to \mathbb{N}$ to find the count of each node. Recall that if (u, v) is a node in T_{Σ^2}, then $u \subseteq \mathcal{X}$ and $v \subseteq \mathcal{X}$. The value $J(u, v)$ denotes the number of trajectories (say) $\boldsymbol{x} \in \mathcal{X}^*$ such that $x_1 \in u$ and $x_{|\boldsymbol{x}|} \in v$. In Algorithm 18 we find this mapping efficiently by first calculating $J(w)$ for all $w \in \mathcal{L}(T_{\Sigma^2})$. Then, we use the fact that the count of an internal node is the total of the count of all its children (Equation 3.3) to find the count of all internal nodes.

Algorithm 19 Estimating start and end state pairs

```
1: Input: The discretized space $\Sigma$, the dataset of trajectories $F : \Sigma^* \to \mathbb{N}$, a hierarchical partition
   $S$ of $\mathcal{X}$, the privacy budget parameter $\varepsilon_3$, and the threshold $\theta$.
2: Output: $P \in \Sigma \times \Sigma$, a set with estimated start and end state pairs for trajectories.

3: // Constructing input for PrivTree.
4: Use Algorithm 18 with the input $\Sigma$, $F$, and $S$ to find $T_{\Sigma^2}$ and $J$.

5: // Calling the PrivTree algorithm.
6: Use PrivTree$_{T_{\Sigma^2}}$ (Algorithm 5) with the input $J$, $\varepsilon_3/2$, and $\theta$ to find a subtree $S_{\Sigma^2}$.

7: // Laplace Mechanism.
8: Sample $\eta(w)$ for all $w \in \mathcal{L}(S_{\Sigma^2})$ as independent random variables of Lap$\left(\frac{2}{\varepsilon_3}\right)$.
9: Set $m(w) \leftarrow \max\left\{\lfloor J(v) + \eta(w) \rfloor, 0\right\}$ for all $w \in \mathcal{L}(S_{\Sigma^2})$.

10: // Sampling.
11: Initialize an empty set $P$.
12: for $w = (u, v) \in \mathcal{L}(S_{\Sigma^2})$ do
13:     Add $m(w)$ random elements, sampled independently with repetition from $\mathcal{L}\left(S|_u\right) \times \mathcal{L}\left(S|_v\right)$,
    to $P$.
```

5.5.4.2. PrivTree *to find the start and end state pairs.* We use the PrivTree$_{T_{\Sigma^2}}$ algorithm (Algorithm 5) to find a subtree S_{Σ^2}. Note that the notation PrivTree$_{T_{\Sigma^2}}$ signifies that the we are looking for a subtree of T_{Σ^2}. In this case, we use the input J, $\varepsilon_3/2$, and θ. We use only half the privacy budget available here since we will use the other half to find the count at the leaves of S_{Σ^2}. $\theta \geq 0$ is again a hyperparameter, which we discuss later in experiments.

The next part is to find an estimated count of trajectories starting and ending at the leaves of S_{Σ^2}. We use the remaining $\varepsilon_3/2$ privacy budget and the true counts as per the mapping J. This is a histogram query and we use the Laplace Mechanism to find this count.

Finally, we have to sample the start and end pairs. Let us use the notation $S|_u$ to denote the subtree of S rooted at u. Then, $\mathcal{L}\left(S|_u\right)$ is the set of all leaves in S for which u is an ancestor. For any node $w = (u, v)$ in $\mathcal{L}\left(S^2_\Sigma\right)$ we first find the set (say) $Q = \mathcal{L}\left(S|_u\right) \times \mathcal{L}\left(S|_v\right)$ which contains all possible pairs from $\mathcal{L}(S) \times \mathcal{L}(S)$ whose start and end pairs are contained in w. Then, we sample $m(w)$ random elements in Q. We do this process for all leaf nodes $w \in \mathcal{L}\left(S^2_\Sigma\right)$ and keep adding the samples to P. Thus the output of Algorithm 19, $P \subseteq \Sigma \times \Sigma$ contains the estimated start and end pairs for the trajectories.

THEOREM 5.5.3 (Privacy of start and end pair sampling). *Algorithm 19 satisfies ε_3-differential privacy.*

PROOF. Let us first focus on the output of Algorithm 18. Note that the input hierarchical partition tree S is not considered to be private information. Hence, the construction of T_{Σ^2} which only depends on S is trivially differentially private. Let f and $\tilde{f}$ be any pair of neighboring datasets as per Definition 2.1.1. Let J and $\tilde{J}$ be their corresponding extension to $V\left(T_{\Sigma^2}\right)$ then it follows that $\left\|J - \tilde{J}\right\|_{C(T)} = 1$.
Note that the construction of S_{Σ^2} and m in Algorithm 19 is done simply by an application of PrivTree and the Laplace Mechanism respectively. Both of these steps take a privacy budget $\varepsilon_3/2$ each. Hence by Theorem 3.3.1 and Theorem 2.2.4 they both satisfy $\varepsilon_3/2$ differential privacy. Thus the algorithm so far is ε_3-differentially private. The sampling is simply a post-processing of the output of these steps and does not violate the privacy guarantees. Hence Algorithm 19 satisfies ε_3-differential privacy. □

5.5.5. Sampling trajectories. Once we have generated P, the set of all pairs of cells in Σ where the trajectories start and end, we finally move to sample trajectories. As discussed in Section 5.3.7 our trajectory-generating process is a random walk conditioned on start and end states. We provide our algorithm formally in Algorithm 20.

5.5.5.1. *Self-loops.* Before we discuss the sampling process further, we need to address a key issue of *self-loops* in the discretized trajectory. Indeed, by its very nature, the process of discretization is likely to introduce many self-loops in the trajectories. However, we will avoid self-loops when generating trajectories as conditioned Markov chains. Thus we will reduce the length of trajectories to generate. Below are the two reasons we believe this length reduction helps our sampling process.

- Recall that, to generate a trajectory of length t which is conditioned on some end state, we need to find the k-step transition probabilities for all $k = 1, 2, \ldots, t-1$. This computation will be very taxing for large values of t, and thus reducing the length reduces the sampling time complexity.
- Intuitively, if the start and end states are fixed, a longer Markov chain will have more randomness than a shorter one.

Algorithm 20 Sampling trajectories

```
1: Input: The discretized space $\Sigma$, the adjacency graph with estimated transition probabilities
   $\hat{\mathcal{G}} = (\Sigma, E, \hat{w})$, and a set of start and end cells of the trajectories $P \in \Sigma \times \Sigma$.
2: Output: $g : \mathcal{X}^* \to \mathbb{N}$, a synthetic dataset of trajectories in $\mathcal{X}$.

3: // Length of the most probable path.
4: Set $\mu(e) \leftarrow \Big(-\ln\big(w(e)\big)\Big)$ for all $e \in E$.
5: for unique pairs $(u, v) \in P$ do
6:     Find shortest path $\boldsymbol{\sigma}$ between $u$ and $v$ in the graph $(\Sigma, E, \mu)$ using the Dijkstra's algorithm.
7:     Set $t_{mode}(u, v) \leftarrow |\boldsymbol{\sigma}|$.

8: // Conditioned random walk.
9: Initialize $G : \Sigma^* \to \mathbb{N} \leftarrow 0$.
10: Initialize $g : \mathcal{X}^* \to \mathbb{N} \leftarrow 0$.
11: for $(u, v) \in P$ do
12:     Sample $t$ from the exponential distribution with scale $\big(\ln 2/t_{mode}(u, v)\big)$.
13:     Sample a trajectory $\boldsymbol{\sigma} \in \Sigma_t$ as a Markov chain $X_1, X_2, \ldots, X_t$ conditioned on $X_1 = u$ and
        $X_t = v$, using the transition graph $\hat{\mathcal{G}}$.

14:     // Add self-loops.
15:     Set $\boldsymbol{\sigma'}$ as an empty sequence.
16:     for $i = 1, 2, \ldots, t$ do
17:         repeat
18:             $\boldsymbol{\sigma'} \leftarrow \boldsymbol{\sigma'} \oplus \sigma_i$.
19:         until a sample from Bernoulli $\big(\hat{w}(\sigma_i, \sigma_i)\big)$ is 0.

20:     // Add to synthetic dataset of trajectories in $\Sigma$.
21:     Set $G(\boldsymbol{\sigma'}) \leftarrow G(\boldsymbol{\sigma'}) + 1$.

22:     // Convert to trajectory in $\mathcal{X}$.
23:     Set $\boldsymbol{x}$ as empty sequence.
24:     for $i = 1, 2, \ldots, \big|\boldsymbol{\sigma'}\big|$ do
25:         Sample a random point $x_i \in \sigma_i$.
26:         Set $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus x_i$.
27:     Set $g(\boldsymbol{x}) \leftarrow g(\boldsymbol{x}) + 1$.
```

Once we have generated a trajectory as conditioned Markov chain, then we reiterate through all the states in the trajectory and add self-loops. Steps 14 through 19 in Algorithm 20 contain our process our adding self-loops. Here the symbol $\oplus$ denotes an operator for concatenating a value to a sequence. And Bernoulli (p) is a random variable with Bernoulli distribution with probability of success p.

5.5.5.2. *Find the length of trajectory.* A key requirement for finding the probability of transitions for such a Markov chain is the length of the trajectory. Let us assume that we are looking for the length of a trajectory for some fixed pair of start and end cells. Inspired from [33] we assume that the trajectory length follows an exponential distribution with parameter $\ln 2/m$, where m is the median length of all such trajectories. However, to find this median length from the input data, we will have to spend some privacy budget. To avoid spending any more budget, and to only use the differentially private quantities generated so far, instead of true median, we use an estimate of the mode of the length of trajectories.

Moreover, since we want to avoid self-loops in the conditioned Markov chain, we are looking for the most probable path without self-loop. Let $(\sigma_1, \sigma_2, \ldots, \sigma_t)$ be any path between the states $\sigma_1 = u$ and $\sigma_t = v$. Then, it follows that,

$$\begin{aligned}\mathbb{P}\Big\{\big((X_2, \ldots, X_t) = (\sigma_2, \ldots, \sigma_t)\big) \mid X_1 = \sigma_1\Big\} &= \prod_{i=2}^{t} \hat{w}(\sigma_{i-1}, \sigma_i) \\ &= \exp\left(-\sum_{i=2}^{t}\Big(-\ln\big(\hat{w}(\sigma_{i-1}, \sigma_i)\big)\Big)\right)\end{aligned}$$

Hence, we first convert the weight of edges from $\hat{w}$ to μ such that for any edge $e \in E$, $\mu(e) = -\ln(\hat{w}(e))$. Then, the shortest path between the nodes u and v in the graph $\hat{\mathcal{G}}(\Sigma, E, \mu)$ is the most probable path from u to v. To find such shortest path we will use the popular Dijkstra's algorithm [18]. We use the length of this path when calculating the scale of exponential distribution for the length of trajectories.

The rest of the steps in the Algorithm are very self-explanatory. Moreover, since we do not use the input data at any step of this algorithm, it is trivially differentially private.

5.5.6. Overall Algorithm. We finally provide our complete algorithm in Algorithm 21. Note that if we set $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$, then Algorithm 21 satisfies ε-differential privacy by Sequential composition (Theorem 2.1.3).

5.6. Experiments and results

5.6.1. Experiment setting. In this section, we present some preliminary results for producing a differentially private synthetic trajectory dataset using our algorithm. We compare our proposed method against AdaTrace[1] [34] as the baseline, run with the default settings. We use the Geolife dataset [2] in these experiments. Moreover, we restrict to trajectories that only contain

[1]https://github.com/git-disl/AdaTrace

Algorithm 21 Overall algorithm: synthetic trajectory generation

```
1: Input: Trajectory dataset $f : \mathcal{X}^* \to \mathbb{N}$, a hierarchical decomposition $T$ of $\mathcal{X}$, the privacy
   budget parameters $\varepsilon_1, \varepsilon_2, \varepsilon_3$, the sensitivity distribution $\alpha$, and the threshold count $\theta$.
2: Output: $g : \mathcal{X}^* \to \mathbb{N}$, a synthetic dataset of trajectories in $\mathcal{X}$.

3: // Discretization.
4: Get $S, \Sigma$ from Algorithm 16 with input $f, T, \varepsilon_1, \theta$.
5: Extend $f$ to $F : \Sigma^* \to \mathbb{N}$ using Equation (5.3).

6: // Adjacency graph.
7: Set $E \subseteq \Sigma \times \Sigma$ with $(\sigma, \sigma') \in E$ if $\sigma'$ is adjacent to $\sigma$ as per Definition 5.3.1.
8: Set $w : E \to [0, 1]$ calculated as per Equation (5.6).
9: Set $\mathcal{G} \leftarrow (\Sigma, E, w)$.
10: Get $\Sigma$ from Algorithm 16 with input $f, T, \varepsilon_1, \theta$.

11: // Transition probabilities.
12: Get estimated probability of transitions, $\hat{w}$, from Algorithm 17 with input $F, \mathcal{G}, \varepsilon_2, \alpha$.
13: Set $\hat{\mathcal{G}} \leftarrow (\Sigma, E, \hat{w})$.

14: // Start and end state pairs.
15: Get $P$, a set of start and end states for trajectories to generate, from Algorithm 19 with input
    $\Sigma, F, S, \varepsilon_3, \theta$.

16: // Sampling.
17: Get synthetic dataset $g : \mathcal{X}^* \to \mathbb{N}$ from Algorithm 20 with input $\Sigma, \hat{\mathcal{G}}, P$.
```

points with latitude and longitude in the range $(116.15, 116.60)$ and $(39.75, 40.10)$ respectively. This results in a dataset with $15,583$ trajectories and a total of $13,151,175$ points. Let ε be the overall privacy budget. Then, we split the budget as $\varepsilon_1 = 0.3\varepsilon$, $\varepsilon_2 = 0.4\varepsilon$, and $\varepsilon_3 = 0.3\varepsilon$.

Next, we discuss the various hyper-parameters involved. For discretization, we use Algorithm 16 with $\beta = 2$ as the fanout (the maximum number of children of any node) and $\theta = 0$ as the count threshold. These choice of β and θ are inspired by the experiments in Section 3.6.8 of Chapter 3. We use both uniform and exponential decay, as presented in Equation (5.10), for the sensitivity distribution α in Algorithm 17.

Similar to experiments in Section 3.6.3 of Chapter 3, the experiments were performed on a personal device with 8 cores of 3.60GHz 11th Generation Intel Core i7 processor and 32 GB RAM. Moreover, since the dataset is based on taxi locations in the city of Beijing, we use the road network of Beijing as our underlying space $\mathcal{X}$. We implement all algorithms in Python and to use the road network as geometry we rely on the spatial computing libraries GeoPandas [43], Shapely [31], NetworkX [35], and OSMnx [9]. These libraries help us efficiently perform discretization from $\mathcal{X}$ to Σ and its inverse after sampling trajectories.

5.6.2. Metrics. In this section, we present the metrics used to demonstrate the proposed method's utility empirically. Our metrics are similar to what has been used in previous works [34, 37, 70]. For some of the metrics to be meaningful and tractable, we need to discretize the space $\mathcal{X}$. In most of the previous works such as [33, 34, 37] a uniform grid of size 6×6 is used for discretization when measuring the metrics. We would like to underscore that it is a very coarse grid and the results on this grid are likely not representative of real-world utility. For example, the area of the urban Beijing city, as per [72] is roughly 6400 sq mi and a uniform grid with 36 cells implies that each cell has an area of 180 sq mi. For comparison, Denver city has an area of about 155 sq mi [73]. [70] have used a finer grid of size 20×20, that is 400 cells. In our metrics we use an even finer grid of size 40×40, that is 1600 cells which results in each cell capturing a small area of about 4 sq mi.

In the description of the metrics, we use $f : \mathcal{X}^* \to \mathbb{N}$ and $g : \mathcal{X}^* \to \mathbb{N}$ as the true and synthetic trajectory datasets respectively. Σ represents a partition of $\mathcal{X}$ created using a uniform grid with 1600 cells. Using Eqution (5.3), we transform f and g to $F : \Sigma^* \to \mathbb{N}$ and $G : \Sigma^* \to \mathbb{N}$ respectively such that they are the trajectories in Σ. With these notations, we now discuss the metrics.

5.6.2.1. *Start and end pair distribution.* This metric measures the error in the distribution of start and end pair locations of the discretized trajectories. Following the notations used in Section 5.3.6, let $\mathcal{C}_{se}$ and $\hat{\mathcal{C}}_{se}$ be the functions that represent the number of trajectories starting and ending at a pair of cells in Σ. We calculate them using Equation (5.7).

The first metric we use is the average relative error (ARE) in the value of the function $\hat{C}_{se}$ as compared to C_{se}. We represent this metric as *SE_ARE* and define it in Equation (5.11). Here, the notation $supp(C_{se}) \subset \Sigma^2$ is represents the support of the function C_{se}, that is, $supp(C_{se}) = \left\{(\sigma, \sigma') \in \Sigma^2 \mid C_{se}(\sigma, \sigma') > 0\right\}$. Similarly $supp(\hat{C}_{se})$ can be defined. Thus, *SE_ARE* calculates the error only over the pair of cells for which either the true or the synthetic trajectories have at least one entry. We use $\alpha = 1$ in our calculations.

(5.11)

$$SE\_ARE \coloneqq \frac{1}{\left|\left(supp(C_{se}) \cup supp(\hat{C}_{se})\right)\right|} \sum_{(\sigma,\sigma') \in \left(supp(C_{se}) \cup supp(\hat{C}_{se})\right)} \left(\frac{\left|C_{se}(\sigma,\sigma') - \hat{C}_{se}(\sigma,\sigma')\right|}{\max\left\{C_{se}(\sigma,\sigma'), \alpha\right\}} \right).$$

The second metric we use is the Jensen-Shanon metric [2] which is the square root of the Jensen-Shanon divergence [53] on given probability distributions. We represent this as a function JSD in this section. Note that we assume that JSD will normalize the distributions if needed and also ensure that the support for the two distributions is the same. Finally, we calculate our metric SE_JSD as the value of JSD on the two empirical distributions given by C_{se} and $\hat{C}_{se}$, and define it in Equation (5.12) below

$$SE\_JSD \coloneqq JSD\left(C_{se}, \hat{C}_{se}\right). \tag{5.12}$$

5.6.2.2. *Frequent patterns.* This metric measures the error in the occurrence of frequent patterns in the discretized trajectories. We only measure this error for the top-k patterns in the true dataset that have lengths between 2 and l. Similar to [70], in our experiments, we use $k = 200$ and $l = 5$. Following the notations used in Section 5.3.3, let $\mathcal{C}_{subtrajectories}(\boldsymbol{\sigma}, F)$ represent the total number of occurrences of $\boldsymbol{\sigma}$ as a subtrajectory of trajectories in F, as defined in Equation (5.5).
We first, find the set of k patterns (subtrajectories) of length between 2 and l with the most occurrence in F. Let $\Sigma^*_{k,l}$ denote this set, and we define it as,

$$\Sigma^*_{k,l} \coloneqq \underset{Q \subseteq \left(\Sigma^2 \cup \cdots \cup \Sigma^l\right), |Q|=k}{\arg\max} \left(\sum_{\boldsymbol{\sigma} \in Q} \mathcal{C}_{subtrajectories}(\boldsymbol{\sigma}, F) \right). \tag{5.13}$$

Then, similar to SE_ARE and SE_JSD, we define FP_ARE and FP_JSD respectively, but restricted to $\Sigma^*_{k,l}$. We define them formally as,

$$FP\_ARE \coloneqq \frac{1}{\left|\Sigma^*_{k,l}\right|} \sum_{\boldsymbol{\sigma} \in \Sigma^*_{k,l}} \left(\frac{\left|\mathcal{C}_{subtrajectories}(\boldsymbol{\sigma}, F) - \hat{\mathcal{C}}_{subtrajectories}(\boldsymbol{\sigma}, G)\right|}{\mathcal{C}_{subtrajectories}(\boldsymbol{\sigma}, F)} \right), \tag{5.14}$$

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.jensenshannon.html

Table 5.1. Comparing various metrics between the proposed method and AdaTrace [34] for synthetic trajectories generation.

ε	Method	Sensitivity distribution	SE_ARE	SE_JSD	FP_ARE	FP_JSD	QE
1.0	AdaTrace	-	**1.12**	0.8	1.0	0.73	**0.19**
1.0	Proposed	Exponential	1.18	**0.63**	**0.96**	0.39	0.23
1.0	Proposed	Uniform	1.19	0.64	0.98	**0.26**	0.27
2.0	AdaTrace	-	1.17	0.8	1.0	0.73	**0.18**
2.0	Proposed	Exponential	**1.14**	**0.6**	0.97	0.32	0.21
2.0	Proposed	Uniform	1.17	**0.6**	**0.96**	**0.27**	0.26

and,

$$FP\_JSD \coloneqq JSD\left(\left(C_{subtrajectories}(\boldsymbol{\sigma}, F)\right)_{\boldsymbol{\sigma}\in\Sigma^*_{k,l}}, \left(\hat{C}_{subtrajectories}(\boldsymbol{\sigma}, G)\right)_{\boldsymbol{\sigma}\in\Sigma^*_{k,l}}\right). \tag{5.15}$$

5.6.2.3. *Trajectory density.* Our third metric compares the number of trajectories passing through a subset of the space $\mathcal{X}$ in the true vs synthetic dataset. We measure this metric using *queries*, which are functions represented as subsets of $\mathcal{X}$ such that for any query q we define,

$$q(f) \coloneqq \sum_{\boldsymbol{x}\in\mathcal{X}^*} f(x)\cdot \mathbb{1}_{|\boldsymbol{x}\cap q|>0}. \tag{5.16}$$

Then, for a given set of queries Q and a constant $\alpha > 0$, we measure our metric Query error (QE) as,

$$QE \coloneqq \frac{1}{|Q|}\sum_{q\in Q}\left(\frac{\left|q(f)-q(g)\right|}{\max\left\{q(f),\alpha\right\}}\right). \tag{5.17}$$

Note that this metric does not require the space to be discretized. However, the query set is important and has a role similar to the discretization grid such that if we want to measure this metric in high resolution, we need queries with a small area. Indeed, in our experiments, we use 500 randomly generated rectangular queries, each with an area in the range 0.01 to 0.1 percent of the area of $\mathcal{X}$. We use α as 0.1 percent of $|f|$. This setting is inspired from [79].

5.6.3. Results. We present the results of our experiments in Table 5.1. Our experiments suggest that the proposed method outperforms AdaTrace on metrics related to start-end pair distribution and frequent patterns. We do not see a better performance in terms of the query error, which needs to be investigated further. The results also suggest that the sensitivity distribution with exponential decay is typically better than the uniform distribution. However, in these experiments we have not extensively explored various parameters in our proposed method such as privacy budget distribution among steps and the max allowed depth of the hierarchical decomposition. Further experimentation will help to conclude the utility of the proposed method compared to existing works.

CHAPTER 6

Conclusion and Future Work

In this dissertation, we focused on the problem of generating synthetic datasets under the privacy guarantees of differential privacy. In particular, we looked at datasets that have been generated over time. We presented methods for spatial and tabular streaming release of synthetic data as well as for generating one-time trajectory datasets. In this section, we briefly summarize these contributions and discuss the open problems.

6.1. Summary

In Chapters 3 and 4 we consider streams with only a few points being contributed by a user. In Chapter 3 we presented a method for differentially private and streaming synthetic data generation for spatial (and other low-dimensional) spaces. We complement the work with empirical results and demonstrate the utility of our method on real-world and simulated datasets.

In Chapter 4 we build on the techniques discussed in Chapter 3 together with existing methods for offline generation of tabular data to provide a method for differentially private and streaming synthetic data generation for high-dimensional tabular datasets. We present a baseline method together with its proof of accuracy and compare our proposed method with this baseline via experiments.

In Chapter 5 we look at datasets where the user may contribute a large number of points. To reduce the complexity of the problem we limit to generating trajectories, for a one-time release and over low-dimensional data space. We identify that many existing methods consider a very coarse grid for discretization of the trajectories which is a key component of the method. We further illustrate the problems associated with using a high-resolution discretization and address these problems in our proposed method. We demonstrate the utility of our method with some preliminary experiments.

6.2. Open directions

In this dissertation, we restricted our attention to streams where a user may contribute only a few points throughout time. Hence, we did not explicitly model the correlation between various points a user may contribute over time. The problem of generating a privacy-preserving stream of high-dimensional synthetic data without restricting the number of contributions a user can make remains open.

A simpler problem is to generate one-time synthetic trajectories in a high-dimensional space. We believe that this problem has also not been explored in existing works. For example, a key step in many existing works (and our proposed method) is discretization which would not scale to high-dimensional trajectory datasets.

Bibliography

[1] *Eviction Notices — DataSF — City and County of San Francisco — data.sfgov.org*. https://data.sfgov.org/Housing-and-Buildings/Eviction-Notices/5cei-gny5/about_data. [Accessed 25-01-2024].

[2] *GeoLife GPS Trajectories*. Microsoft Store - Download Center, Available at https://www.microsoft.com/en-us/download/details.aspx?id=52367.

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy*, in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 308–318.

[4] J. Abowd, D. Kifer, S. L. Garfinkel, and A. Machanavajjhala, *Census TopDown: Differentially Private Data, Incremental Schemas, and Consistency with Public Knowledge*, tech. rep., 2019.

[5] M. Andrés, N. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, *Geo-Indistinguishability: Differential Privacy for Location-Based Systems*, in Proceedings of the ACM Conference on Computer and Communications Security, Nov. 2013.

[6] S. Aydore, W. Brown, M. Kearns, K. Kenthapadi, L. Melis, A. Roth, and A. A. Siva, *Differentially Private Query Release Through Adaptive Projection*, in Proceedings of the 38th International Conference on Machine Learning, vol. 139 of Proceedings of Machine Learning Research, PMLR, July 2021, pp. 457–467.

[7] C. Bauer and C. Strauss, *Location-based advertising on mobile devices*, Management Review Quarterly, 66 (2016), pp. 159–194.

[8] B. Becker and R. Kohavi, *Adult*, 1996. Published: UCI Machine Learning Repository.

[9] G. Boeing, *OSMnx: A Python package to work with graph-theoretic OpenStreetMap street networks*, Journal of Open Source Software, 2 (2017), p. 215. Publisher: The Open Journal.

[10] M. Bun, M. Gaboardi, M. Neunhoeffer, and W. Zhang, *Continual Release of Differentially Private Synthetic Data from Longitudinal Data Collections*, Proceedings of the ACM on Management of Data, 2 (2024), pp. 94:1–94:26.

[11] M. Bun and T. Steinke, *Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds*, in Theory of Cryptography, Berlin, Heidelberg, 2016, Springer Berlin Heidelberg, pp. 635–658.

[12] T.-H. H. Chan, E. Shi, and D. Song, *Private and Continual Release of Statistics*, ACM Transactions on Information and System Security, 14 (2011), pp. 26:1–26:24.

[13] Y. Chen, A. Machanavajjhala, M. Hay, and G. Miklau, *PeGaSus: Data-Adaptive Differentially Private Stream Processing*, in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, New York, NY, USA, Oct. 2017, Association for Computing Machinery, pp. 1375–1388.

[14] E. Cho, S. A. Myers, and J. Leskovec, *Friendship and Mobility: User Movement in Location-Based Social Networks*, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, New York, NY, USA, 2011, Association for Computing Machinery, pp. 1082–1090. event-place: San Diego, California, USA.

[15] A. Cohen, *Attacks on Deidentification's Defenses*, in 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, Aug. 2022, USENIX Association, pp. 1469–1486.

[16] T. Cunningham, G. Cormode, H. Ferhatosmanoglu, and D. Srivastava, *Real-world trajectory sharing with local differential privacy*, Proceedings of the VLDB Endowment, 14 (2021), pp. 2283–2295.

[17] D. Desfontaines and B. Pejó, *SoK: Differential privacies*, Proceedings on Privacy Enhancing Technologies, 2020 (2019), pp. 288 – 313.

[18] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[19] B. Ding, J. Kulkarni, and S. Yekhanin, *Collecting Telemetry Data Privately*, in Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.

[20] B. Donovan and D. Work, *New york city taxi trip data (2010-2013)*, 2016.

[21] M. Douriez, H. Doraiswamy, J. Freire, and C. T. Silva, *Anonymizing NYC Taxi Data: Does It Matter?*, in 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2016, pp. 140–148.

[22] C. Dwork, *Differential privacy*, in Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33, Springer, 2006, pp. 1–12.

[23] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, *Differential privacy under continual observation*, in Symposium on the Theory of Computing, 2010.

[24] C. Dwork, M. Naor, O. Reingold, and G. N. Rothblum, *Pure Differential Privacy for Rectangle Queries via Private Partitions*, in Advances in Cryptology – ASIACRYPT 2015, vol. 9453, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 735–751. Series Title: Lecture Notes in Computer Science.

[25] C. Dwork, A. Smith, T. Steinke, and J. Ullman, *Exposed! A Survey of Attacks on Private Data*, Annual Review of Statistics and Its Application, 4 (2017), pp. 61–84. Publisher: Annual Reviews Type: Journal Article.

[26] K. El Emam, E. Jonker, L. Arbuckle, and B. Malin, *A Systematic Review of Re-Identification Attacks on Health Data*, PLOS ONE, 6 (2011), pp. 1–12. Publisher: Public Library of Science.

[27] U. Erlingsson, V. Pihur, and A. Korolova, *RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response*, in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, New York, NY, USA, 2014, Association for Computing Machinery, pp. 1054–1067. event-place: Scottsdale, Arizona, USA.

[28] M. Fanaeepour and B. I. P. Rubinstein, *Histogramming Privately Ever After: Differentially-Private Data-Dependent Error Bound Optimisation*, in 2018 IEEE 34th International Conference on Data Engineering (ICDE), Apr. 2018, pp. 1204–1207. ISSN: 2375-026X.

[29] M. Gaboardi, E. J. G. Arias, J. Hsu, A. Roth, and Z. S. Wu, *Dual Query: Practical Private Query Release for High Dimensional Data*, in Proceedings of the 31st International Conference on Machine Learning, PMLR, June 2014, pp. 1170–1178. ISSN: 1938-7228.

[30] S. Garfinkel, J. M. Abowd, and C. Martindale, *Understanding database reconstruction attacks on public data*, Communications of the ACM, 62 (2019), pp. 46–53. Publisher: ACM New York, NY, USA.

[31] S. Gillies, C. van der Wel, J. Van den Bossche, M. W. Taves, J. Arnott, B. C. Ward, and others, *Shapely*, Jan. 2023.

[32] A. Guha Thakurta and A. Smith, *(Nearly) Optimal Algorithms for Private Online Learning in Full-information and Bandit Settings*, in Advances in Neural Information Processing Systems, vol. 26, Curran Associates, Inc., 2013.

[33] M. E. Gursoy, L. Liu, S. Truex, and L. Yu, *Differentially Private and Utility Preserving Publication of Trajectory Data*, IEEE Transactions on Mobile Computing, 18 (2019), pp. 2315–2329.

[34] 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, CCS '18, New York, NY, USA, Oct. 2018, Association for Computing Machinery, pp. 196–211.

[35] A. A. Hagberg, D. A. Schult, and P. J. Swart, *Exploring Network Structure, Dynamics, and Function using NetworkX*, in Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, 2008, pp. 11 – 15.

[36] M. Hardt, K. Ligett, and F. Mcsherry, *A Simple and Practical Algorithm for Differentially Private Data Release*, in Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.

[37] 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, 8 (2015), pp. 1154–1165.

[38] Y. He, R. Vershynin, and Y. Zhu, *Online Differentially Private Synthetic Data Generation*, Feb. 2024. arXiv:2402.08012 [cs, math, stat] version: 1.

[39] J. Henriksen-Bulmer and S. Jeary, *Re-identification attacks—A systematic literature review*, International Journal of Information Management, 36 (2016), pp. 1184–1192.

[40] P. Jain, I. Kalemaj, S. Raskhodnikova, S. Sivakumar, and A. Smith, *Counting Distinct Elements in the Turnstile Model with Differential Privacy under Continual Observation*, in Advances in Neural Information Processing Systems, vol. 36, Curran Associates, Inc., 2023, pp. 4610–4623.

[41] K. Jiang, D. Shao, S. Bressan, T. Kister, and K.-L. Tan, *Publishing trajectories with differential privacy guarantees*, in Proceedings of the 25th International Conference on Scientific and Statistical Database Management, SSDBM '13, New York, NY, USA, 2013, Association for Computing Machinery. event-place: Baltimore, Maryland, USA.

[42] F. Jin, W. Hua, M. Francia, P. Chao, M. E. Orlowska, and X. Zhou, *A Survey and Experimental Study on Privacy-Preserving Trajectory Data Publishing*, IEEE Transactions on Knowledge and Data Engineering, 35 (2023), pp. 5577–5596.

[43] K. Jordahl, J. V. d. Bossche, M. Fleischmann, J. Wasserman, J. McBride, J. Gerard, J. Tratner, M. Perry, A. G. Badaracco, C. Farmer, G. A. Hjelle, A. D. Snow, M. Cochran, S. Gillies, L. Culbertson, M. Bartos, N. Eubank, maxalbert, A. Bilogur, S. Rey, C. Ren, D. Arribas-Bel, L. Wasser, L. J. Wolf, M. Journois, J. Wilson, A. Greenhall, C. Holdgraf, Filipe, and F. Leblanc, *geopandas/geopandas: v0.8.1*, July 2020.

[44] J. Jordon, J. Yoon, and M. v. d. Schaar, *PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees*, Sept. 2018.

[45] M. Joseph, A. Roth, J. Ullman, and B. Waggoner, *Local Differential Privacy for Evolving Data*, in Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018.

[46] P. Kairouz, B. Mcmahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu, *Practical and Private (Deep) Learning Without Sampling or Shuffling*, in Proceedings of the 38th International Conference on Machine Learning, vol. 139 of Proceedings of Machine Learning Research, PMLR, July 2021, pp. 5213–5225.

[47] J. S. Kim, Y. D. Chung, and J. W. Kim, *Differentially Private and Skew-Aware Spatial Decompositions for Mobile Crowdsensing*, Sensors (Basel, Switzerland), 18 (2018), p. 3696.

[48] J. W. Kim, K. Edemacu, J. S. Kim, Y. D. Chung, and B. Jang, *A Survey Of differential privacy-based techniques and their applicability to location-Based services*, Computers & Security, 111 (2021), p. 102464.

[49] G. Kumar, T. Strohmer, and R. Vershynin, *An Algorithm for Streaming Differentially Private Data*, Jan. 2024. arXiv:2401.14577 [cs, math, stat].

[50] J. Lee, I. Shin, and G.-L. Park, *Analysis of the Passenger Pick-Up Pattern for Taxi Location Recommendation*, in 2008 Fourth International Conference on Networked Computing and Advanced Information Management, vol. 1, Sept. 2008, pp. 199–204.

[51] M. Li, L. Zhu, Z. Zhang, and R. Xu, *Achieving differential privacy of trajectory data publishing in participatory sensing*, Information Sciences: an International Journal, 400 (2017), pp. 1–13.

[52] X. Li, F. Tramer, P. Liang, and T. Hashimoto, *Large language models can be strong differentially private learners*, arXiv preprint arXiv:2110.05679, (2021).

[53] J. Lin, *Divergence measures based on the Shannon entropy*, IEEE Transactions on Information Theory, 37 (1991), pp. 145–151. Conference Name: IEEE Transactions on Information Theory.

[54] T. Liu, G. Vietri, and S. Z. Wu, *Iterative Methods for Private Synthetic Data: Unifying Framework and New Methods*, in Advances in Neural Information Processing Systems, vol. 34, Curran Associates, Inc., 2021, pp. 690–702.

[55] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, *L-diversity: privacy beyond k-anonymity*, in 22nd International Conference on Data Engineering (ICDE'06), Apr. 2006, pp. 24–24. ISSN: 2375-026X.

[56] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala, *Optimizing Error of High-Dimensional Statistical Queries Under Differential Privacy*, Journal of Privacy and Confidentiality, 13 (2023). Number: 1.

[57] R. McKenna, G. Miklau, and D. Sheldon, *Winning the NIST Contest: A scalable and general approach to differentially private synthetic data*, Journal of Privacy and Confidentiality, 11 (2021). Number: 3.

[58] R. Mckenna, D. Sheldon, and G. Miklau, *Graphical-model based estimation and inference for differential privacy*, in Proceedings of the 36th International Conference on Machine Learning, vol. 97 of Proceedings of Machine Learning Research, PMLR, June 2019, pp. 4435–4444.

[59] I. Mironov, *Rényi Differential Privacy*, 2017 IEEE 30th Computer Security Foundations Symposium (CSF), (2017), pp. 263–275.

[60] A. Narayanan and V. Shmatikov, *Robust De-anonymization of Large Sparse Datasets*, in 2008 IEEE Symposium on Security and Privacy (sp 2008), May 2008, pp. 111–125. ISSN: 2375-1207.

[61] L. Ni, C. Li, X. Wang, H. Jiang, and J. Yu, *DP-MCDBSCAN: Differential Privacy Preserving Multi-Core DBSCAN Clustering for Network User Data*, IEEE Access, 6 (2018), pp. 21053–21063. Conference Name: IEEE Access.

[62] W. Qardaji, W. Yang, and N. Li, *Differentially private grids for geospatial data*, in 2013 IEEE 29th International Conference on Data Engineering (ICDE), Apr. 2013, pp. 757–768. ISSN: 1063-6382.

[63] J. Sarathy and S. Vadhan, *Analyzing the Differentially Private Theil-Sen Estimator for Simple Linear Regression*, arXiv preprint arXiv:2207.13289, (2022).

[64] X. Sun, Q. Ye, H. Hu, Y. Wang, K. Huang, T. Wo, and J. Xu, *Synthesizing Realistic Trajectory Data With Differential Privacy*, IEEE Transactions on Intelligent Transportation Systems, 24 (2023), pp. 5502–5515.

[65] L. Sweeney, *k-anonymity: A model for protecting privacy*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10 (2002), pp. 557–570. Publisher: World Scientific.

[66] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, *Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12*, Sept. 2017. arXiv:1709.02753 [cs].

[67] Y. Tao, R. McKenna, M. Hay, A. Machanavajjhala, and G. Miklau, *Benchmarking Differentially Private Synthetic Data Generation Algorithms*, Feb. 2022. arXiv:2112.09238 [cs].

[68] D. P. Team, *Learning with Privacy at Scale*, tech. rep., Apple, Dec. 2017.

[69] R. Torkzadehmahani, P. Kairouz, and B. Paten, *Dp-cgan: Differentially private synthetic data and label generation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019, pp. 0–0.

[70] H. Wang, Z. Zhang, T. Wang, S. He, M. Backes, J. Chen, and Y. Zhang, *{PrivTrace}: Differentially Private Trajectory Synthesis by Adaptive Markov Models*, 2023, pp. 1649–1666.

[71] T. Wang, J. Q. Chen, Z. Zhang, D. Su, Y. Cheng, Z. Li, N. Li, and S. Jha, *Continuous Release of Data Streams under both Centralized and Local Differential Privacy*, in Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21, New York, NY, USA, Nov. 2021, Association for Computing Machinery, pp. 1237–1253.

[72] Wikipedia, *Beijing*. `http://en.wikipedia.org/w/index.php?title=Beijing&oldid=1220781080`, Apr. 2024. Page Version ID: 1220781080.

[73] ———, *List of United States cities by area*. `http://en.wikipedia.org/w/index.php?title=List%20of%20United%20States%20cities%20by%20area&oldid=1213735488`, Mar. 2024. Page Version ID: 1213735488.

[74] ———, *Weather (Apple)*. `https://en.wikipedia.org/w/index.php?title=Weather_(Apple)&oldid=1220543391`, Apr. 2024. Page Version ID: 1220543391.

[75] C. Xu, J. Ren, D. Zhang, Y. Zhang, Z. Qin, and K. Ren, *GANobfuscator: Mitigating information leakage under GAN via differential privacy*, IEEE Transactions on Information Forensics and Security, 14 (2019), pp. 2358–2371. Publisher: IEEE.

[76] 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, WWW '17, Republic and Canton of Geneva, CHE, 2017, International World Wide Web Conferences Steering Committee, pp. 1241–1250. event-place: Perth, Australia.

[77] X. Yue, H. A. Inan, X. Li, G. Kumar, J. McAnallen, H. Sun, D. Levitan, and R. Sim, *Synthetic text generation with differential privacy: A simple and practical recipe*, arXiv preprint arXiv:2210.14348, (2022).

[78] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, *PrivBayes: Private Data Release via Bayesian Networks*, ACM Transactions on Database Systems, 42 (2017), pp. 25:1–25:41.

[79] J. Zhang, X. Xiao, and X. Xie, *PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions*, in Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, New York, NY, USA, June 2016, Association for Computing Machinery, pp. 155–170.

[80] M. Zhang, J. Liu, Y. Liu, Z. Hu, and L. Yi, *Recommending Pick-up Points for Taxi-drivers Based on Spatio-temporal Clustering*, in 2012 Second International Conference on Cloud and Green Computing, Nov. 2012, pp. 67–72.