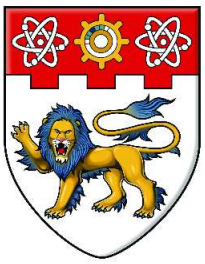


NANYANG
TECHNOLOGICAL
UNIVERSITY

Algorithms for Synthetic Data Release under Differential Privacy

A thesis submitted
for a degree of Doctor of Philosophy
by

Jun ZHANG

School of Computer Engineering
Nanyang Technological University

Jan 2016

Abstract

Releasing sensitive data while preserving privacy is an important problem that has attracted considerable attention in recent years. The state-of-the-art paradigm for addressing this problem is differential privacy, which offers a strong degree of privacy protection without making restrictive assumptions about the adversary. Most efforts to date to perform differentially private data release end up mired in complexity, overwhelm the signal with noise, and are not effective for use in practice. In this thesis, we introduce three novel solutions for complex data publication under differential privacy, namely, PrivBayes, PrivTree and the ladder framework. Compared to the previous work, our methods (i) enable the private release of a wide range of data types, i.e., multi-dimensional tabular data, spatial data, sequence data and graph data, (ii) improve the utility of released data by introducing significantly less perturbations in data modelling and (iii) are query-independent, such that many different queries (linear or non-linear) can be accurately evaluated on the same set of released data.

First of all, we present PrivBayes, a differentially private method for releasing multi-dimensional tabular data. Given a dataset D, PrivBayes first constructs a Bayesian network $\mathcal{N}$, which (i) provides a succinct model of the correlations among the attributes in D and (ii) allows us to approximate the distribution of data in D using a set $\mathcal{P}$ of low-dimensional marginals of D. After that, PrivBayes injects noise into each marginal in $\mathcal{P}$ to ensure differential privacy, and then uses the noisy marginals and the Bayesian network to construct an approximation of the data distribution in D. Finally, PrivBayes samples tuples from the approximate distribution to construct a synthetic dataset, and then releases the synthetic data. Intuitively, PrivBayes circumvents the curse of dimensionality, as it injects noise into the low-dimensional marginals in $\mathcal{P}$ instead of the full-dimensional dataset D. Private construction of Bayesian networks turns out to be significantly challenging, and we introduce a novel approach that uses a surrogate function for mutual information to build the model more accurately. We experimentally evaluate PrivBayes on real data, and demonstrate that it significantly outperforms existing solutions in terms of accuracy.

Second, we introduce PrivTree, a differentially private algorithm for releasing spatial and sequential datasets. Given a set D of tuples defined on a domain Ω, PrivTree constructs a histogram over Ω to approximate the tuple distribution in D. It adopts a hierarchical decomposition approach, which recursively splits Ω into sub-domains and

computes a noisy tuple count for each sub-domain, until all noisy counts are below a certain threshold. Previous efforts based on hierarchical decomposition require that we (i) impose a limit h on the recursion depth in the splitting of Ω and (ii) set the noise in each count to be proportional to h. The choice of h is a serious dilemma: a small h makes the resulting histogram too coarse-grained, while a large h leads to excessive noise in the tuple counts used in deciding whether sub-domains should be split. PrivTree completely eliminates its dependency on a pre-defined h. The rationale behind is a novel mechanism that (i) exploits a new analysis on the Laplace distribution and (ii) enables us to use only a constant amount of noise in deciding whether a sub-domain should be split, without worrying about the recursion depth of splitting. We demonstrate the application of PrivTree in modelling spatial data, and show that it can be extended to handle sequence data (where the decision in sub-domain splitting is not based on tuple counts but a more sophisticated measure). Our experiments on a variety of real datasets show that PrivTree considerably outperforms the states of the art in terms of data utility.

Last, we investigate the problem of protecting the privacy of individuals in graph structured data, and propose a differentially private solution, namely, the ladder framework. It specifies a probability distribution over possible outputs that is carefully defined to maximize the utility for the given input, while still providing the required privacy level. The distribution is designed to form a 'ladder', so that each output achieves the highest 'rung' (maximum probability) compared to less preferable outputs. We show how our ladder framework can be applied to problems of counting the number of occurrences of subgraphs, a vital objective in graph analysis, and give algorithms whose cost is comparable to that of computing the count exactly. Our experimental study confirms that our method outperforms existing methods for counting triangles and stars in terms of accuracy, and provides solutions for some problems for which no effective method was previously known. The results of our algorithms can be used to estimate the parameters of suitable graph models, allowing synthetic graphs to be sampled and released.

Acknowledgments

I would like to express my sincere appreciation to my supervisor Dr. Xiaokui Xiao for his professional guidance and constant support throughout my Ph.D. study. It has been a wonderful experience working with Dr. Xiao in the past years, and I have greatly benefited from his insightful comments and feedback. I am also very thankful for his help with my writing and presentation skills through the years.

I would also like to acknowledge my coauthors and colleagues. They include Dr. Xin Cao, Dr. Peilin Zhao, Dr. Diwen Zhu, Dr. Ning Chen, Dr. Wenqing Lin, Dr. Zongyang Ma, Youze Tang, Sibo Wang (Nanyang Technological University), Dr. Zhenjie Zhang, Dr. Yin Yang, Prof. Marianne Winslett (Advanced Digital Sciences Center, Singapore), Prof. Graham Cormode (University of Warwick), Dr. Cecilia M. Procopiuc (Google Inc.), Dr. Divesh Srivastava (AT&T Labs - Research) and Dr. Xing Xie (Microsoft Research).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The advancement of information technologies has made it never easier for various organizations (e.g., hospitals, bus companies, census bureaus) to create large repositories of user data (e.g., patient data, passenger commute data, census data). Such data repositories are of tremendous research value – for example, statistical analysis of patient data can help evaluate health risks, determine best practices, and develop new treatments; passenger commute data provides invaluable insights into the effectiveness of transportation systems; census data is an essential source of information for demographic research. Despite of their research values, those data repositories can seldom be accessed for public studies, due to concerns over individual privacy. Recent years have witnessed quite a few incidents in which organizations fail to protect privacy in the release of sensitive data, e.g., the AOL search log scandal in 2006 [Barbaro and Zeller, 2006] and the de-anonymization of Netflix prize data in 2007 [Narayanan and Shmatikov, 2008].

A common practice to address this issue is to anonymize the data, i.e., to modify the data and eliminate the information that may lead to privacy intrusion. The simplest form of data anonymization is to remove all personal identifiers (such as names and IDs). Nevertheless, recent research [Backstrom et al., 2007; Calandrino et al., 2011; Narayanan et al., 2012; Narayanan and Shmatikov, 2008, 2009; Srivatsa and Hicks, 2012; Sweeney, 2002] has shown that removing personal identifiers alone is insufficient for privacy protection, since the remaining attributes in the data may still be exploited to re-identify an individual. For instance, in late 1990s, Massachusetts' Group Insurance Commission released a table of medical records of 135,000 state employees, after removing sensitive information like names, addresses and social security numbers. Latanya Sweeney, then an MIT grad student, re-identified Governor William Weld's health data by correlating his birthdate and zip code with public voting records [Sweeney, 2002]; Anthony Tockar of Neustar Research, showed how celebrities get exposed by a public dataset released by the

New York City Taxi and Limousine Commission (TLC). By Googling news and gossip reports about celebrities taking cabs in NYC, and correlating them with the data from the TLC, Tockar found out exactly where and when in New York celebrities climbed into cabs, where they traveled in those cabs and even how much they paid; Narayanan and Shmatikov [2009] give a concrete demonstration of how their de-anonymization algorithm works by applying it to Flickr and Twitter, two large, real-world online social networks. They show that a third of the users who are verifiable members of both Flickr and Twitter can be recognized in the completely anonymous Twitter graph with only 12% error rate. The concerns over individual privacy have motivated numerous anonymization techniques (see [Dwork, 2010; Sarwate and Chaudhuri, 2013; Fung et al., 2010] for surveys) that aim to provide better privacy protection. The existing techniques, however, still offer inadequate privacy assurance, because (i) they rely on restrictive assumptions on how a malicious user may attack the data, whereas (ii) those assumptions can be easily violated in practice [Elliot et al., 2000].

Motivated by the practical importance of data anonymization and the deficiencies of the existing methods, the current state-of-the-art solution is to seek the ε-differential privacy [Dwork et al., 2006b], due to its strong privacy guarantees and robustness against adversaries with background knowledge. Given a sensitive dataset, ε-differential privacy requires that any information published from the data must be randomly perturbed to prevent inference of sensitive information. In particular, it ensures that, even if a potential adversary knows the exact details of all but one record in the data, it would still be difficult for the adversary to infer the information about the remaining record. A vast body of research on differential privacy has been published in recent years [Dwork, 2006; McSherry and Talwar, 2007; Ding et al., 2011; Barak et al., 2007; Blum et al., 2008; Chaudhuri et al., 2011; Cormode et al., 2012a,b; Lei, 2011; Zhang et al., 2012, 2013]; see also the survey [Dwork, 2008] for a summary of this topic.

1.2 Problems and Motivations

Putting differential privacy into practice remains a challenging problem. Since its proposal, there have been many efforts to develop differentially private mechanisms and algorithms, for different kinds of input databases and for different objectives for the end use. In this thesis, we investigate an important problem that is frequently encountered in differential privacy literature: given the input dataset D, how to release a manipulated version of D that (i) satisfies the requirements of ε-differential privacy and (ii) retains the statistical characteristics of input data as much as possible. The significance of this problem is that the released dataset is multi-purpose in that it can support a large set of analysis tasks simultaneously. Existing solutions on differential privacy, however, cannot

effectively handle the publication of complex data, despite that many important types of real data (e.g., census data, medical records, GPS trajectory data) are complex in nature. In what follows, we specify the difficulties in modelling four different categories of complex data, namely, multi-dimensional tabular data, spatial data, sequential data and graph data.

Multi-dimensional Tabular Data. The canonical case in differentially private data publication is when the database can be modeled as a table, where each row may contain information about an individual (say, details of their medical status, or employment information). Then, the aim is to release some manipulated version of this information so that this can still be used for the intended purpose, but the privacy of individuals in the database is preserved. However, most existing techniques encounter difficulties when trying to release even moderately high dimensional data – that is, an input table with half a dozen columns or more. The reasons for these problems are two-fold:

1. *Output Scalability*: Most algorithms (see, e.g., [Xiao et al., 2011]) either explicitly or implicitly represent the database as a vector x of size equal to the *domain size*, that is, the product of cardinalities of the attributes. For many natural data sets, the domain size m is orders of magnitude larger than the data size n [Cormode et al., 2012b]. Hence, these algorithms become inapplicable for any realistic dataset with a moderate-to-high number of attributes. For example, a million row table with ten attributes, each of which has 20 possible values, results in a domain size (and hence an output size) of $m = 20^{10} \approx 10$TB, which is very unwieldy and slow to use compared to the input which can be measured in megabytes.
2. *Signal-to-noise Ratio*: When the multi-dimensional database is represented as a vector x, the average count in each entry, given by n/m, is typically very small. Once noise is added to x to obtain another vector x^*, the noise completely dominates the original signal, making the published vector x^* next to useless. For example, if the table above has size $n = 1$M, the average entry count is $n/m = 10^{-7}$. By contrast, the average noise injected to achieve, e.g., differential privacy with parameter $\varepsilon = 0.1$ has expected magnitude around 10. Even if the data is skewed in the domain space, i.e., there are some entries $x[i]$ with high counts, such peaks are infrequent and so the vast majority of published values $x^*[i]$ are useless.

Spatial and Sequential Data. We put these two types of data together as their private releases share the same pattern: Given a set D of tuples defined over a domain Ω, we aim to decompose Ω into a set S of sub-domains and publish a noisy count of the tuples contained in each sub-domain, such that S and the noisy counts approximate the tuple distribution in D as accurately as possible. Take the spatial data as an example.

The domain of spatial data (i.e., a region that defines the range of data points) can be decomposed into a set of disjoint sub-regions, each of which is associated with the number of data points located in it. By adding sufficient noise to each count, we are able to obtain a private summary of the original data distribution [Cormode et al., 2012a; Qardaji et al., 2013a].

To find an appropriate decomposition, the prior art mostly adopts a hierarchical approach, which (i) recursively splits Ω into sub-domains and computes a noisy tuple count for each of them, and (ii) stops splitting a sub-domain when its noisy count is smaller than a threshold. This approach, albeit intuitive, requires a pre-defined limit h on the maximum depth of recursion when splitting Ω. The reason is that, to ensure differential privacy, the amount of noise injected in each tuple count has to be proportional to the maximum recursion depth, and hence, h must be fixed in advance so that the algorithm can decide the correct noise amount to use.

Nevertheless, the choice of h is a serious dilemma: for the algorithm to produce fine-grained sub-domains of Ω, h cannot be small; yet, increasing h would lead to noisier tuple counts, and thus more errors in deciding whether a sub-domain should be split. This makes it difficult to select an appropriate h that could result in an accurate approximation of the input data. Furthermore, we cannot tune h directly on the input dataset; otherwise, the choice of h itself reveals private information and violates differential privacy. To mitigate these issues, existing work relies on heuristics to select an appropriate value of h, and to generate fine-grained decompositions even when h is small. However, those heuristics are rather ineffective when the input data follows a skewed distribution (which is often the case in practice).

Graph Data and Subgraph Counts. A large amount of private data can be well-represented by graphs, e.g, social network activities, communication patterns, disease transmission, and movie preferences. However, applying differential privacy to graph data has proven much more fraught with difficulties than other forms of data. The main technical roadblock is that direct application of standard methods seems to require such a large scale of random modification of the graph input as to completely lose all its properties. Concretely, any method which tries to directly output a modified version of the input graph under differential privacy is indistinguishable from random noise.

Instead, progress has been made by focusing on releasing properties of the input graph under differential privacy, rather than the graph itself. For example, it is straightforward to release the number of nodes and edges with this guarantee, by appealing to standard techniques. It is valuable to find statistical properties of the graph, since there are many random graph models which take these as parameters (e.g., Kronecker graph models [Leskovec et al., 2010; Mir and Wright, 2012] and Exponential Random Graph Models [Karwa et al., 2014]), and allow us to sample synthetic graphs from this family.

The properties of the graph are also important in their own right, determining, for instance, measures of how much clustering there is in the graph, and other characteristics of group behavior. See the discussion in [Karwa et al., 2014], and references therein.

The problem then becomes one of providing effective ways to release statistics on the input graph privately. Some important statistics are subgraph counts: the number of occurrences of particular small subgraphs within the input graph. This remains a challenging problem, since standard differential privacy techniques still add a large amount of noise to the result. Take as a canonical problem the question of counting the number of triangles within a graph. The presence or absence of one relationship (represented by an edge) in the graph can contribute to a large number of potential triangles, and so the noise added to the count has to be scaled up by this amount. There have been numerous prior attempts to privately release statistics on graphs via more complex methods, but these still suffer from high noise, and high running time.

1.3 Approaches and Contributions

To address the above problems that are unsolvable using existing solutions, we propose three novel frameworks for differentially private data release, namely, PrivBayes [Zhang et al., 2014] , PrivTree [Zhang et al., 2016] and the ladder framework [Zhang et al., 2015].

PrivBayes for Multi-dimensional Tabular Data. We first introduce PrivBayes, an effective solution to the problem of publishing differentially private multi-dimensional data. Unlike the bulk of prior work, which focuses on optimizing the output for specific workloads (e.g., range queries, cube queries), we aim to approximate the full-dimensional distribution of the original data with a data-dependent set of well-chosen low-dimensional distributions, in the belief that, for a sufficiently accurate approximation, the resulting data will maintain high accuracy for almost any type of (linear or non-linear) query. By working in low-dimensional spaces, we avoid the signal-to-noise problem. Although we compare to the full-dimensional distribution for evaluation purposes, our approach never needs to compute this explicitly, thus avoiding the scalability problem.

To achieve this goal, we start from the well-known *Bayesian network* model, which is widely studied in the statistical and machine learning communities [Koller and Friedman, 2009]. Bayesian networks combine low-dimensional distributions to approximate the full-dimensional distribution of a data set, and are a simple but powerful example of a graphical model. Our algorithm consists of the following steps:

1. *Network Learning*: We describe how to compute a differentially private Bayesian network that approximates the full-dimensional distribution via the exponential mechanism. This step requires new theoretical insights, which are described in Section 3.2.

We improve on this basic approach by defining a new quality function for use in the exponential mechanism, which results in significantly better networks being found. The definition and computation of this function are one of our main technical contributions; see Section 3.3.

2. *Distribution Learning*: We explain how to compute the necessary differentially private distributions of the data in the subspaces of the Bayesian network, via the Laplace Mechanism.
3. *Data synthesis*: We show how to generate synthetic data from the differentially private Bayesian network, without explicitly materializing the global distribution.

In Section 3.5, we provide an extensive experimental evaluation of the accuracy of the synthetic datasets generated above, over workloads of linear and non-linear queries. In each case, we compare to prior methods specifically designed to optimize the accuracy for that type of workload. Our experiments show that PrivBayes is often more accurate than any prior method, even though it is not optimized for any specific type of query. When PrivBayes is less accurate than some prior method, the accuracy loss is small and, we believe, an acceptable tradeoff, since PrivBayes offers a generic solution that does not require prior knowledge of the workload and works well on many different types of queries.

PrivTree for Spatial and Sequential Data. The second solution we present is PrivTree, an algorithm for publishing spatial and sequential data with differential privacy guarantees. It adopts the hierarchical approach but completely eliminates the dependency on a pre-defined h. In particular, PrivTree requires only *a constant amount of noise* in deciding whether a sub-domain should be split, which enables it to generate fine-grained decompositions without worrying about the recursion depth. Such a surprising improvement is obtained with a novel mechanism for differential privacy that exploits a non-trivial analysis on the Laplace noise [Dwork et al., 2006b] to derive an extremely tight privacy bound. Its central insight is that, in the context of hierarchical decomposition, it is possible to publish a sequence S of 0/1 values using $O(1)$ noise, regardless of the sensitivity of S [Dwork et al., 2006b]. In contrast, the standard Laplace mechanism requires that noise amount must be proportional to S's sensitivity.

To demonstrate the applications of PrivTree, we apply it to the private modeling of spatial data, and present a non-trivial extension to tackle sequence data, for which we adopt an advanced Markov model and utilize a sophisticated measure (instead of tuple counts) to decide whether a sub-domain should be split. We experimentally evaluate our algorithms on a variety of real data, and show that they considerably outperform the states of the art in terms of data utility. In addition, we present an in-depth analysis on the connection between PrivTree and the sparse vector technique [Hardt, 2011; Dwork

and Roth, 2013; Lee and Clifton, 2014], a technique widely adopted for data publishing under differential privacy. We show that there exists a variant of sparse vector technique [Lee and Clifton, 2014] that could have been used to implement PrivTree, if its privacy guarantees are as claimed in previous work [Lee and Clifton, 2014]. Nevertheless, we prove that the sparse vector technique variant does not satisfy differential privacy, which makes it inapplicable in our context.

In summary, we make the following contributions: (i) we propose PrivTree, a differentially private algorithm for hierarchical decomposition that eliminates the dependency on a pre-defined threshold of the recursion depth; (ii) we present applications of PrivTree in modeling spatial and sequence data (Sections 4.2 and 4.3); (iii) we analyze the connection between PrivTree and the sparse vector technique, and point out a misclaim about the latter in [Lee and Clifton, 2014] (Section 4.4); and (iv) we conduct extensive experiments to demonstrate the superiority of PrivTree over the states of the art (Section 4.5).

The Ladder Framework for Graph Data and Subgraph Counts. Last, we show how the ladder framework answers subgraph counting queries in a differentially private manner [Dwork, 2006], and uses the released counts to generate synthetic graphs. We focus on a number of important classes of subgraphs. The function $f_{\triangle}$ counts the number of triangles in the graph: this is the most heavily studied subgraph counting query, as the triangle is the smallest non-trivial subgraph of interest. Other important subgraph counting functions include: $f_{k\star}$, for counting k-stars (one node of degree k connected to k nodes of degree 1); $f_{k\mathbb{C}}$, for counting k-cliques; and $f_{k\triangle}$ for counting k-triangles (a pair of adjacent nodes i, j that have k common neighbors).

Our ladder framework achieves the following list of criteria that are desired for differentially private subgraph counting:

- It should be applicable to a wide range of queries, e.g., triangle counting, k-star counting, k-triangle counting and k-clique counting for a constant k;
- Its time complexity should be no larger than that of computing the true query answer for the queries of interest;
- It should have high accuracy on its output, bettering that of any previous applicable solution.

This relies on the careful design of a probability distribution for outputting answers to count queries that tries to maximize the probability of outputting the true answer, or one that is near to it, and minimize the probability of outputting answers that are far from the true answer. The framework of differential privacy places constraints on how these probabilities should behave: the probability of providing output x for input graph g

should be "close" (as determined by the parameter ε) to that for input graph g', if g and g' are close (differ by one edge). Rather than go right back to first principles to design these probability distributions, we make use of the exponential mechanism [McSherry and Talwar, 2007], since this approach is general: any differentially private mechanism can be expressed as an exponential mechanism.

Instantiating the exponential mechanism still takes some careful design. We arrive at the design of a symmetric "ladder function" specific to the input graph g, so-called because it is formed from a series of rungs. The first rung groups together a number of outputs that are close to the true answer to $f_H(g)$ (i.e. $f_H(g) \pm 1, f_H(g) \pm 2 \ldots$). The second rung groups together outputs that are a little further away, and so on. The height of each rung determines the probability of outputting the corresponding values, while the width is the number of output values sharing the same height. Ideally, we would make each rung as narrow as possible, to maximize the dropoff in probabilities away from f_H. However, to guarantee differential privacy, we need to make the rungs wide enough so that the ith rung of the ladder for g overlaps with the $i-1$st, ith or $i+1$st rung of the ladder for all neighboring g' (in the language of the exponential mechanism, we seek bounded 'sensitivity' of the derived quality function). We call this the "small leap property".

It turns out that we can design such ladder functions which have rungs that are not too wide but which satisfy the small leap property. Moreover, these can be computed efficiently from the input graph g and used to perform the required output sampling. Our development of these objects takes multiple steps. First, we fill in the necessary details of differential privacy (Section 5.1). Then we show how the use of ladder functions can lead to a practical differentially private solution, if we assume the small leap property, and that the functions converge to a constant rung width (Section 5.2). We still have to show that we can create such functions, and this is the focus of Section 5.2.1.1, where we show that the concept of "local sensitivity at distance t" can be used to create a ladder function with the required properties. This provides a general solution which satisfies privacy; however, some of the quantities required can be time consuming to compute for certain subgraph counting queries. Hence, Section 5.3 considers how to instantiate our framework for specific queries, and provides alternate ladder functions for some queries to enable efficient computation. Our experimental evaluation of this approach, and comparison with previous works (where applicable) is detailed in Section 5.4.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 surveys the state-of-art methods in differential privacy. Chapter 3 presents PrivBayes. Chapter 4 presents PrivTree. Chapter 5 presents the ladder framework. Chapter 6 concludes the work we have finished and briefly introduces our future work.

Chapter 2

Literature Survey

In this chapter, we introduce the basic concepts that we rely on throughout this thesis. We start with the formal definitions of differential privacy in Section 2.1, and then present two fundamental solutions, namely, the Laplace mechanism and the exponential mechanism, in Section 2.2. Last, we conduct extensive surveys on prior work and relevant techniques to aggregate queries in Sections 2.3, learning tasks in Section 2.4 and graph statistics in Section 2.5, respectively.

2.1 Differential Privacy

Let D be a sensitive dataset with n tuples, and $\mathcal{A}$ be a data publishing algorithm that takes D as input and releases a set of information $\mathcal{A}(D)$. Differential privacy requires that $\mathcal{A}(D)$ should only rely on the general properties of D, and should not be sensitive to the presence or absence of any particular tuple in the data. This ensures that, when an attacker observes $\mathcal{A}(D)$, he/she would not be able to infer much about any individual tuple in the original data, i.e., privacy is preserved. More formally, differential privacy is defined based on the concept of neighboring datasets, as shown in the following.

Definition 2.1 (Neighboring Datasets [Dwork et al., 2006b]) *Two datasets are neighboring if one of them can be obtained by inserting a tuple into the other.*

Definition 2.2 (ε-Differential Privacy [Dwork et al., 2006b]) *A randomized algorithm $\mathcal{A}$ satisfies ε-differential privacy, if for any two neighboring datasets D and D' and for any possible output O of $\mathcal{A}$,*

$$\Pr\big[\mathcal{A}(D) = O\big] \leq e^{\varepsilon} \cdot \Pr\big[\mathcal{A}(D') = O\big], \tag{2.1}$$

where $\Pr[\cdot]$ *denotes the probability of an event.*

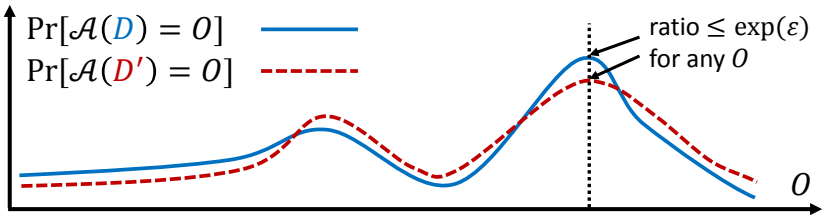


Figure 2.1: An illustration of ε-differential privacy

The ε-differential privacy guarantees the hardness for inferring any information of any tuple in the input dataset. This hardness is controlled by the parameter ε, referred to as the *privacy budget*. Generally, a smaller privacy budget indicates stronger privacy protection, but also noisier results. Figure 2.1 illustrates the definition of ε-differential privacy.

Moreover, it is worth mentioning that there exists a relaxed version of ε-differential privacy, namely, (ε, δ)-differential privacy.

Definition 2.3 ((ε, δ)-Differential Privacy [Nissim et al., 2007]) *A randomized algorithm $\mathcal{A}$ satisfies ε-differential privacy, if for any two neighboring datasets D and D' and for any set of outputs S of $\mathcal{A}$, we have*

$$\Pr\left[\mathcal{A}(D) \in S\right] \leq e^{\varepsilon} \cdot \Pr\left[\mathcal{A}(D') \in S\right] + \delta.$$

Under this privacy notion, a randomized algorithm is considered privacy preserving if it achieves ε-differential privacy with a high probability (decided by δ). This relaxed notion is useful in the scenarios where ε-differential privacy is too strict to allow any meaning results to be released (see [Korolova et al., 2009; Götz et al., 2012] for examples). As we will show later, however, all data release problems studied in this thesis can be conducted effectively under ε-differential privacy, i.e., we do not need to resort to (ε, δ)-differential privacy to achieve meaningful results.

2.2 Fundamental Approaches

By Definition 2.2, it is easy to know that a deterministic algorithm cannot possibly satisfy ε-differential privacy for any value of ε. Hence, it must be randomly perturbed to meet the privacy requirement. Two fundamental approaches for this purpose are the Laplace mechanism [Dwork et al., 2006b] and the exponential mechanism [McSherry and Talwar, 2007]. The Laplace mechanism considers a function f that takes D as input and outputs a vector of real numbers, and it aims to release $f(D)$ with differential privacy. To achieve

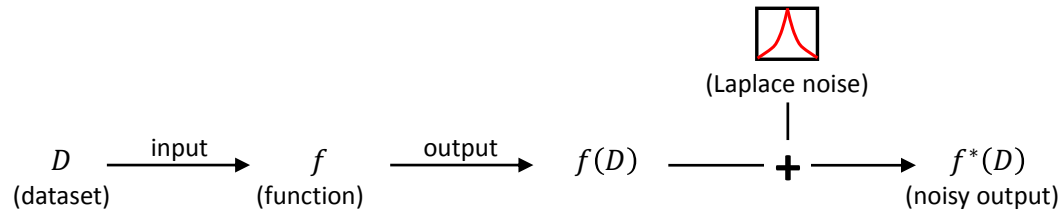


Figure 2.2: An illustration of the Laplace mechanism

this objective, it adds i.i.d. noise into each output value of f, such that the noise η follows a Laplace distribution with the following probability density function:

$$\Pr[\eta = x] = \frac{1}{2\lambda} e^{-|x|/\lambda}. \tag{2.2}$$

We denote the above distribution as $\text{Lap}(\lambda)$, and refer to λ as the *scale* (since the standard deviation of $\text{Lap}(\lambda)$ is proportional to λ). Figure 2.2 illustrates the workflow of the Laplace mechanism.

Dwork et al. [2006b] prove that the Laplace mechanism achieves ε-differential privacy if $\lambda \geq S(f)/\varepsilon$, where $S(f)$ is the *sensitivity* of function f:

Definition 2.4 (Sensitivity [Dwork et al., 2006b]) *Let f be a function that maps a dataset into a fixed-size vector of real numbers. The sensitivity of f is defined as*

$$S(f) = \max_{D,D'} \|f(D) - f(D')\|_1 , \tag{2.3}$$

where $\|\cdot\|_1$ denotes the L_1 norm, and D and D' are any two neighboring datasets.

Intuitively, $S(f)$ measures the maximum possible change in f's output when we modify one arbitrary tuple in f's input. A large $S(f)$ indicates that f may reveal significant information about a certain tuple, in which case we should inject a large amount of noise into fs output to protect privacy. This explains why the Laplace mechanism sets the standard deviation of the noise proportional to $S(f)/\varepsilon$.

For an analysis task with a categorical output (e.g., an ID), injecting random noise no longer yields meaning results. The exponential mechanism tackles this problem by performing random perturbations during the selection of the output. Specifically, given the input data D and the set Ω of all possible output values, the user assigns each possible output $\omega \in \Omega$ a *quality score* $q(D, \omega)$; higher scores correspond to better values. Then, the exponential mechanism computes the probability for selecting each possible output value in the domain, and selects one randomly based on these probabilities. In particular, the probability for choosing a value $\bar{\omega}$ satisfies:

$$\Pr\left[\bar{\omega} \text{ is selected}\right] \propto \exp\left(\frac{\varepsilon \cdot q(D, \bar{\omega})}{2 \max_{\omega \in \Omega} S(q(\cdot, \omega))}\right), \tag{2.4}$$

where $S(q(\cdot, \omega))$ is the sensitivity of $q(\cdot, \omega)$ as a function of the input database, defined in Equation (2.3), i.e., the maximum possible difference of $q(\cdot, \omega)$ for any two neighboring databases. Note that the exponential mechanism does not make any assumptions on the quality function q.

In short, both the Laplace mechanism and the exponential mechanism can be applied quite generally; however, to be effective we seek to ensure that the noise introduced does not outweigh the signal in the data, and that it is computationally efficient to apply the mechanism. This requires a careful design of what functions to use in the mechanisms.

Last, we introduce the composition property of differential private algorithms. It addresses how differential privacy guarantees compose: when accessing the data multiple times via differentially private mechanisms, each of which having its own privacy guarantees, how much privacy is still guaranteed on the union of those outputs?

Lemma 2.1 (Composition Rule [McSherry and Mironov, 2009]) *Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be k algorithms, such that $\mathcal{A}_i$ satisfies ε_i-differential privacy ($i \in [1, k]$). Then, the sequential application of algorithms $(\mathcal{A}_1, \ldots, \mathcal{A}_k)$ on the same dataset satisfies $(\sum_{i=1}^{k} \varepsilon_i)$-differential privacy.*

This lemma is particularly useful in proving that an algorithm ensures differential privacy: we can first decompose the algorithm into a few sequential components, and then analyze each component separately; after that, we can apply Lemma 2.1 to establish the overall privacy guarantee of the algorithm.

2.3 Existing Solutions to Aggregate Queries

In this section, we survey the existing differentially private solutions to a class of aggregate queries, including counts, marginals and histograms. As the most widely studied problem under differential privacy, aggregate queries have attracted considerable attentions in research communities [Barak et al., 2007; Xiao et al., 2011; Ding et al., 2011; Li and Miklau, 2012, 2013; Hardt et al., 2012; Yuan et al., 2012; Yaroslavtsev et al., 2013; Cormode et al., 2012a,b; Qardaji et al., 2013a,b; Su et al., 2015; Chen et al., 2012b,a; Hay et al., 2010; Dwork et al., 2006b; McSherry, 2009]. Answering one single linear counting query under differential privacy is rather simple, as the Laplace mechanism has been proved [Ghosh et al., 2012] to be the solution that introduces the least amount of noise. However, it remains challenging to design the error-optimal solution for a workload of linear counting queries (e.g., a set of marginals). In what follows, we identify some most important techniques in this line of research.

Fourier Transform. Barak et al. [2007] study the problem of releasing marginals along with differential privacy guarantees. The primary contribution is providing a formal guarantee to preserve all the privacy, accuracy, and consistency in the published marginals.

Instead of adding noise to the original data at the cost of accuracy, or adding noise to the published marginals at the cost of consistency, the method proposes (i) transforming the original data into the Fourier domain, (ii) applying the Laplace mechanism on the required Fourier coefficients to achieve differential privacy, and (iii) employing linear programming to obtain a non-negative contingency table based on those noisy Fourier coefficients. One of main problems of this work is that of efficiency, as solving the linear programming could be extremely time-consuming. Moreover, applying Fourier transformation requires all attributes of the data to be binary.

Wavelet Transform. Xiao et al. [2011] investigate the problem of range count answering under differential privacy, and propose Privelet, an effective synopsis for answering arbitrary range counts with a fixed privacy budget. In particular, Privelet makes use of the wavelet transformation of data, and fulfills differential privacy by adding random noise into wavelet coefficients. This addresses range queries, and means that the noise incurred by a range query scales proportionately to the logarithm of its length. This result significantly improves over the Laplace mechanism, as the noise added by Laplace mechanism has to scale linearly to the length of range query. Furthermore, the authors also illustrate how Privelet can be extended to nominal attributes and multi-dimensional data, by adopting novel forms of wavelet transformations.

Data Cubes. Ding et al. [2011] consider the problem of releasing an arbitrary set $\mathcal{L}$ of marginals of input data. The proposed method consists of three steps: (i) choosing an initial subset of $\mathcal{L}$ to compute directly from the input data; (ii) injecting Laplace noise to each selected marginal to preserve privacy; and (iii) computing the remaining marginals in $\mathcal{L}$ from the initial set. However, it turns out to be NP-hard to choose the initial set of $\mathcal{L}$ so that the maximal noise over all published marginals is minimized, or so that the number of marginals with noise below a given threshold is maximized. The authors give approximation algorithms to address this issue. A major limitation is that the running time is exponential in the dimensionality of the domain, making it inapplicable for all but small sets. In addition, accuracy is improved by additional post-processing of the output to restore "consistency" of the counts mandated by the structure (e.g. using the fact that counts in a hierarchy should sum to the count of an ancestor) [Barak et al., 2007; Hay et al., 2010; Ding et al., 2011]; however, this improvement does not overcome the inherent error incurred in high dimensions.

Matrix Mechanism. The aforementioned approaches [Ding et al., 2011; Xiao et al., 2011; Barak et al., 2007] focus on providing the best bounds on noise for a particular set of linear counting queries (e.g., marginals and range count queries). More generally, the matrix mechanism [Li and Miklau, 2012, 2013; Li et al., 2010] and subsequent related approaches [Hardt et al., 2012; Yuan et al., 2012; Yaroslavtsev et al., 2013] take in an

arbitrary workload of counting queries (expressed in terms of a weighted set of inner-product queries), and seek the best strategy that minimizes the noise for these queries. In particular, given a workload, the matrix mechanism requests answers to a different set of queries, called a *query strategy*, which are answered using the standard Laplace mechanism. Noisy answers to the workload queries are then derived from the noisy answers to the strategy queries. This two stage process can result in a more complex correlated noise distribution that preserves differential privacy but increases accuracy. The main challenge of applying the matrix mechanism to practical workloads is to identify the appropriate query strategy, which involves solving a costly semidefinite programming [Li et al., 2010]. Recent years have witnessed quite a few efforts to speeding up the computation of matrix mechanism, including the proposals of the low-rank mechanism [Yuan et al., 2012] and the adaptive mechanism [Li and Miklau, 2013].

Data Reduction Techniques. Some existing approaches are to use data reduction techniques to suppress the scale of noise in released histograms. For example, Cormode et al. [2012b] propose various sampling mechanisms to reduce the size of (and time to produce) the full histogram, while approximately preserving the accuracy of subset-sum queries. However, the accuracy guarantee is with respect to the accuracy of using the entire noisy histogram, rather than the original one, which degrades rapidly with data dimensionality. The approach in [Cormode et al., 2012a] tries to keep the domain size of the histogram small, and the density of data within the new domain high, by adaptively grouping some of the attribute values. This coarser grid representation of the data loses precision, and still leads to small average counts in each grid cell if the dimensionality is large. Cormode et al. [2012a] address the latter problem by using spatial decompositions to define irregular grids over the domain of data, such that the count in each grid cell is sufficiently large. This makes sense for range queries, but requires all attributes to have an ordering. Recently, Chen et al. [2015] also proposed to utilize probabilistic graphical models in modelling sensitive multi-dimensional datasets. They novelly introduced the sparse vector technique [Dwork and Roth, 2013; Hardt, 2011] to the private learning of graphical models, resulting in a significant improvement in utility over previous methods. However, this technique is shown to be flawed (see Section 4.4 and [Chen and Machanavajjhala, 2015]), as its privacy analysis is erroneous. This invalidates the solution proposed in [Chen et al., 2015], and we have not seen any easy way to address this issue. Moreover, Li et al. [2014b] present DPCopula, a data sanitization technique using copula functions for multi-dimensional data. The core of this method is to compute a differentially private copula function along with marginal distributions of each individual dimension, from which the full-dimensional joint distribution can be restored. Though the categorical attributes of small domain size are not well supported, the copula function provides a novel treatment of continuous attributes other than the naive domain

descritization (this is how PrivBayes does). This sheds light on how differential privacy algorithms can be improved in the case when the input dataset contains continuous variables.

Sequential Data Release. Several studies have been proposed to address the issue of publishing sequence databases under differential privacy. In particular, Chen et al. [2012b] propose a differentially private sequence database publishing algorithm that makes use of the prefix tree structure to group sequences with the identical prefix into the same branch. Chen et al. [2012a] employ a variable-length n-gram model to extract information from the sequence databases. The method reduces the dimensionality of the pattern space by restricting the mining on short patterns (n-grams), and then utilize the Markov assumption to reconstruct a sanitized dataset. Moreover, some efforts have been made on differentially private frequent sequence mining. Xu et al. [2015] achieve good utility on mining results by utilizing a sampling-based candidate pruning technique that effectively reduces the number of unpromising subsequences. Bonomi and Xiong [2013] study a slightly different problem: mining frequent consecutive item sequences. They propose a two-phase differentially private algorithm that (i) selects candidate sequences via prefix trees, and then (ii) leverages a database transformation technique to refine the support of candidate sequences.

Other Applications. Last, we summarize some previous work that focuses on applications closely related to aggregate queries. In particular, McSherry [2009] designs the PINQ platform for privacy-preserving data analysis, over a set of aggregate instructions such as count, sum and average. Qardaji et al. [2013a,b] and Su et al. [2015] study the private publication of spatial histograms via hierarchical decompositions. Xu et al. [2012] propose a data-aware solution for private histogram publication. Finally, Li et al. [2014a] present DAWA, the first data- and workload-aware mechanism for answering sets of range queries. The algorithm benefits from the properties of input data as well as the correlations in the workload, which therefore significantly improves over existing competitors.

2.4 Existing Solutions to Learning Tasks

There is also a large class of results that dedicate to applying differential privacy on machine learning tasks, including regression, classification, and clustering. Existing approaches to these problems generally follow the methodology of developing the differentially private version of a commonly used algorithm in the non-private setting. The main challenge faced by this methodology is that the sensitivity of such algorithms (and, consequently, the scale of perturbations) is usually prohibitively high, to the extent that direct use of the Laplace mechanism or the exponential mechanism simply returns noise.

PrivateERM. Chaudhuri et al. [2011] consider differentially private convex empirical risk minimization (PrivateERM), which can be applied to a wide range of optimization problems (e.g., regressions and classifications). The method proposed in [Chaudhuri et al., 2011] provides a new paradigm of objective perturbation, which achieves differential privacy by injecting random noise into the loss function f of the optimization problem. Nevertheless, the correctness of the method relies on some strong assumptions about f, e.g., f must be convex and doubly differentiable, and has a bounded derivative. As a consequence, it cannot handle simple linear regression. [Chaudhuri et al., 2011] demonstrates two specific applications of their method: logistic regression with a non-zero regularization term (logistic regression with zero regularization, however, fails the strongly convex condition), and SVM classification with certain special loss functions such as *Huber loss* (the more commonly used *hinge loss* function is not differentiable [Rubinstein et al., 2012]). Even for these two applications, however, the effectiveness of the method is relatively unstable, and highly sensitive to the choice of the regularization factor. The parameter tuning algorithm in [Chaudhuri et al., 2011] for selecting an appropriate regularization factor consumes a considerable portion of the privacy budget, reducing the overall accuracy of their methods.

PrivateERM Extensions. Rubinstein et al. [2012] investigate differentially private kernel SVMs. Their main focus lies in tackling different types of kernels, including those that involves infinite dimensionality. However, the solutions in [Rubinstein et al., 2012] still assume a standard SVM solver, which incurs high sensitivity and large amounts of noise. Finally, Kifer et al. [2012] improve both the accuracy and the applicability of [Chaudhuri et al., 2011], by giving a tighter bound to the additional error introduced in objective perturbation. Their analysis extends the range of problems to which objective perturbation applies. For example, it allows one to use objective perturbation for convex programs like the Lasso (where the regularizer is non-differentiable) and nuclear norm regularized minimization, which was earlier not possible in [Chaudhuri et al., 2011]. The extension is also critical for applying the objective perturbation technique to the well-known problem of linear regression.

GUPT Framework. GUPT [Mohan et al., 2012] is a general-purpose differentially private analysis system based on the sample-and-aggregate framework [Smith, 2011], which applies to all analysis tasks whose results are not affected by the number of records in the dataset (e.g., AVG is supported, but COUNT and SUM are not). The main idea is to partition the dataset into smaller blocks, and run the analysis algorithm on each of the blocks *without any privacy considerations*. Then, GUPT computes a differentially private average on the results from different blocks. The main strength of GUPT is its general applicability and ease of use. Users with no privacy expertise can conduct differentially private analysis, by simply uploading their data mining programs to GUPT. These come

at a price of low result quality, however, as the differentially private averaging step can still incur high sensitivity for larger output domains. Overall, all existing solutions are limited to the implicit assumption that a model fitting task should be solved using the same algorithm as in the non-private case.

Synthetic Data Release. Another way to conduct complex data analysis under differential privacy is through synthetic data release. Lei [2011] proposes such a data releasing method for regressions that avoids conducting sensitivity analysis directly on the regression outputs. In a nutshell, the method first employs the Laplace mechanism to produce a noisy multi-dimensional histogram of the input data. The granularity of the histogram is carefully optimized for the regression task, in order to preserve the statistical utility in the output. After that, it produces a synthetic dataset that matches the statistics in the noisy histogram, without looking at the original dataset. Finally, it utilizes the synthetic data to compute the regression results. Observe that, the privacy guarantee of this method is solely decided by the procedure that generates the noisy histogram – the subsequent parts of the algorithm only rely on the histogram (instead of the original data), and hence, they do not reveal any information about the input dataset (except for the information revealed by the noisy histogram). This makes it much easier to enforce ε-differential privacy, since the multi-dimensional histogram consists of only counts, which can be processed with the Laplace mechanism in a differentially private manner. Nevertheless, Lei's method [Lei, 2011] is restricted to datasets with small dimensionality. This is caused by the fact that, when the dimensionality of the input data increases, this method would generate noisy histogram with a coarser granularity, which in turn leads to inaccurate synthetic data and regression results.

Other Tasks. Differential privacy has also been applied to other complex data mining tasks. In particular, Li et al. [2012] and Zeng et al. [2012] study privacy-preserving frequent item mining. Friedman and Schuster [2010] investigate private decision trees. Inan et al. [2010] design solutions for record matching under differential privacy. More examples include the work of McSherry and Mironov [2009] to mask the preferences of individual raters in recommendation systems; Rastogi and Nath [2010] to release time-series data based on leading Fourier coefficients; McSherry and Mahajan [2010] for addressing common queries over network trace data; Xiao and Xiong [2015] to model temporal correlations of moving trajectories; Feldman et al. [2009] to release coresets for summarizing geometric data; Gupta et al. [2010] to initiate a systematic study of algorithms for discrete optimization problems in the framework of differential privacy. Finally, Kifer and Machanavajjhala [2011] point out the limitations of differential privacy, and explore alternative privacy definitions [Kifer and Machanavajjhala, 2012].

2.5 Existing Solutions to Graph Statistics

Another bulk of prior work that is highly relevant to this thesis is to release graph statistics with differential privacy guarantees. Hay et al. [2009] first translate the language of differential privacy to the context of graph, and give the formal definitions of edge differential privacy. After that, a plethora of differentially private techniques have been proposed for various graph statistics, especially degree sequences and subgraph counts. Among all existing efforts, an obvious first attempt is to apply the Laplace mechanism [Dwork et al., 2006b]. However, it fails since the sensitivity of most graph queries is very large, making the noise large enough to overwhelm the true answer. In the case of triangle counting, for example, adding or deleting a tuple (which is an edge in graph) can affect a number of triangles proportional to the number of nodes, and hence the sensitivity is very high.

Degree Sequence. Hay et al. [2009] consider the problem of publishing degree sequence, an important summary that is used in many graph models. The method proposed in this work and subsequent related approaches [Karwa and Slavković, 2012; Lin and Kifer, 2013] first utilize the Laplace mechanism to generate a private version of the degree sequence, then apply a postprocessing step to enforce consistency of the sequence in order to reduce noise. Specifically, the theoretical results in [Hay et al., 2009] show that the error scales linearly with the number of unique degrees when the postprocessing step applies, whereas the error with sole Laplace mechanism scales linearly with the number of nodes. It is also worth mentioning that these methods implicitly allow k-star counting, as the the number of k-stars is a function of the degree sequence. Besides the above solutions to first-order degree sequences, Proserpio et al. [2014] and Sala et al. [2011] further extend the problem to higher-order joint degree sequences, which model more detailed information within the sensitive graph. In addition, a recent work by Wang and Wu [2013] improves Sala's solution by calibrating noise to an instance-based measurement, rather than the large sensitivity of queries. Moreover, the authors illustrate that synthetic graphs generated from the noisy joint degree sequence have a series of appealing properties.

Smooth Sensitivity. Regarding the problem of releasing subgraph counts, Nissim et al. [2007] and Karwa et al. [2014] present an idea of utilizing a local version of sensitivity, called *local sensitivity*, to build the noise distribution. The local sensitivity is a function of the input database, which is equal to the maximum change of query answer if one tuple is added in/deleted from the input database. The local sensitivity can be dramatically smaller than its global counterpart, but cannot be used directly to guarantee privacy. Instead, an upper bound on local sensitivity called *smooth sensitivity* is computed as a part of the noise scale. In [Karwa et al., 2014], the authors give algorithms for computing smooth sensitivity of triangle counting $f_{\triangle}$ and k-star counting $f_{k\star}$. However, their idea fails when the smooth sensitivity of a query is hard to compute, e.g., k-triangle counting

$f_{k\triangle}$. In recompense, they propose an approach tailored specifically for $f_{k\triangle}$, which only achieves a restricted version of differential privacy.

Recursive Mechanism. Recursive mechanism [Chen and Zhou, 2013] is a recent approach to releasing subgraph counts with differential privacy. Instead of answering the subgraph counting query directly, the method in [Chen and Zhou, 2013] cleverly releases a lower bound of the query answer, whose sensitivity is relatively low. This method suffers from a bias between the true query answer and the lower bound, in exchange for less noise. The authors claim that their method can be extended to a variety of subgraph counting queries, but it involves solving a linear program with $O(L)$ variables where L equals the number of subgraphs H in the input times the number of edges in H. So the time complexity is super-quadratic in the answer value, making this method infeasible for an input graph of moderate size.

More Applications. Besides the aforementioned approaches, Mir and Wright [2012] show how to bridge differentially private subgraph counts with Kronecker graph models, providing another method to generate synthetic graphs with privacy guarantees. In a different vein, [Hardt and Roth, 2013; Ahmed et al., 2013; Wang et al., 2013] investigate differentially private spectral analysis (e.g., singular value decomposition and random projection) for graph data. Their results imply a great potential to process/release large scale sensitive graphs like the Facebook friendship graph and call/SMS/email networks. Recently, Lu and Miklau [2014] describe the use of a chain mechanism to release alternating k-star and k-triangle counting with an (ε, δ)-differentially private guarantee. They also show how the published counts can be used to estimate the parameters of exponential random graph models.

Beyond Edge Differential Privacy. There is also a branch of work studying other variants of graph differential privacy. For example, [Kasiviswanathan et al., 2013; Blocki et al., 2013; Chen and Zhou, 2013] are the first few methods to provide *node* differential privacy with non-trivial utility guarantees. Node differential privacy is a qualitatively stronger privacy guarantee than edge differential privacy, where two graphs are considered neighbors if they differ in the presence of a node and all its incident edges. As a consequence, it generally requires substantially more noise to protect the released information. Meanwhile, Rastogi et al. [2009] consider a relaxed version of edge differential privacy, called edge *adversarial* privacy, which relies on some assumptions about prior knowledge of the malicious user. As those assumptions might plausibly be violated in practice, this new definition actually provide less privacy assurance, compared to the conventional edge differential privacy.

So far, research community has not reached an agreement on which version of graph differential privacy to use. Though more progress has been made with edge differential privacy, some doubt if it really provides sufficient protection on the database. For

example, Klovdahl et al. [1994] study the social network structure of "a population of prostitutes, injecting drug users and their personal associates." In this graph, an edge represents a sexual interaction or the use of a shared needle. Edges are clearly private information, but so too are other properties like node degree (the number of sexual/drug partners) and even membership in the network. This suggests that node differential privacy might be more plausible in some contexts.

Chapter 3

PrivBayes: Private Release of Tabular Data

This chapter presents PrivBayes, an effective solution to the problem of publishing differentially private multi-dimensional tabular data. Unlike the bulk of prior work, which focuses on optimizing the output for specific workloads (e.g., range queries, cube queries), we aim to approximate the full-dimensional distribution of the original data with a data-dependent set of well-chosen low-dimensional distributions, in the belief that, for a sufficiently accurate approximation, the resulting data will maintain accuracy for a large set of (linear or non-linear) queries. Since our approach is multi-purpose, many different queries can be evaluated on the same set of released data. On the other hand, previous work [Dinur and Nissim, 2003] comes to the conclusion that the more queries a private algorithm wants to support, the worse each answer becomes. This suggests that our approach might be weaker than approaches that target a particular query set; however, we show empirically that the gap is small or non-existent in many natural cases. By working in low-dimensional spaces, we avoid the signal-to-noise problem. Although we compare to the full-dimensional distribution for evaluation purposes, our approach never needs to compute this explicitly, thus avoiding the scalability problem.

To achieve this goal, we start from the well-known *Bayesian network* model, which is widely studied in the statistical and machine learning communities [Koller and Friedman, 2009]. Bayesian networks combine low-dimensional distributions to approximate the full-dimensional distribution of a data set, and are a simple but powerful example of a graphical model. Our algorithm consists of the following steps:

1. (*Network learning*) We describe how to compute a differentially private Bayesian network that approximates the full-dimensional distribution via the exponential mechanism. This step requires new theoretical insights, which are described in Section 3.2. We improve on this basic approach by defining a new quality function for use in the

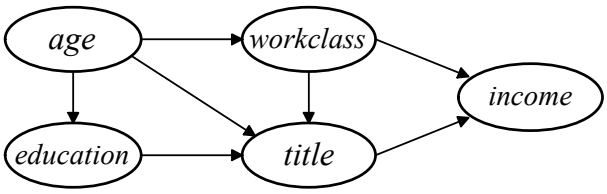


Figure 3.1: A Bayesian network $\mathcal{N}_1$ over five attributes

exponential mechanism, which results in significantly better networks being found. The definition and computation of this function are one of our main technical contributions; see Section 3.3.

2. (*Distribution learning*) We explain how to compute the necessary differentially private distributions of the data in the subspaces of the Bayesian network, via the Laplace Mechanism.

3. (*Data synthesis*) We show how to generate synthetic data from the differentially private Bayesian network, without explicitly materializing the global distribution.

In Section 3.5, we provide an extensive experimental evaluation of the accuracy of the synthetic datasets generated above, over workloads of linear and non-linear queries. In each case, we compare to prior methods specifically designed to optimize the accuracy for that type of workload. Our experiments show that PrivBayes is often *more* accurate than any prior method, even though it is not optimized for any specific type of query. When PrivBayes is less accurate than some prior method, the accuracy loss is small and, we believe, an acceptable tradeoff, since PrivBayes offers a generic solution that does not require prior knowledge of the workload and works well on many different types of queries.

3.1 Bayesian Network

Let $\mathcal{A}$ be the set of attributes on a dataset D, and d be the size of $\mathcal{A}$. A *Bayesian network* on $\mathcal{A}$ is a way to compactly describe the (probability) distribution of the attributes in terms of other attributes. Formally, a Bayesian network is a directed acyclic graph (DAG) that (i) represents each attribute in $\mathcal{A}$ as a node, and (ii) models conditional independence among attributes in $\mathcal{A}$ using directed edges. As an example, Figure 3.1 shows a Bayesian network over a set $\mathcal{A}$ of five attributes, namely, *age*, *education*, *workclass*, *title*, and *income*. For any two attributes $X, Y \in \mathcal{A}$, there exist three possibilities for the relationship between X and Y:

Case 1: direct dependence. There is an edge between X and Y, say, from Y to X. This indicates that for any tuple in D, its distribution on X is determined (in part) by

Table 3.1: The attribute-parent pairs in $\mathcal{N}_1$

i	X_i	Π_i
1	*age*	$\emptyset$
2	*education*	{*age*}
3	*workclass*	{*age*}
4	*title*	{*age*, *education*, *workclass*}
5	*income*	{*workclass*, *title*}

its value on Y. We define Y as a *parent* of X, and refer to the set of all parents of X as its *parent set*. For example, in Figure 3.1, the edge from *workclass* to *income* indicates that the income distribution depends on the type of job (and also on title).

Case 2: weak conditional independence. There exists at least one path (but no edge) between Y and X. Assume without loss of generality that the paths go from Y to X. Then, X and Y are *conditionally independent* given a set $\mathcal{Z}$ that blocks all these paths. For instance, in Figure 3.1, there are four unique paths from *age* to *income*, that are blocked by $\mathcal{Z} = \{workclass, title\}$. This indicates that, given workclass and job title of an individual, her income and age are conditionally independent.

Case 3: strong conditional independence. There is no path between Y and X. Then, X and Y are conditionally independent given any of X's and Y's parent sets.

Formally, a Bayesian network $\mathcal{N}$ over $\mathcal{A}$ is defined as a set of d *attribute-parent pairs*, $(X_1, \Pi_1), \ldots, (X_d, \Pi_d)$, such that

1. Each X_i is a unique attribute in $\mathcal{A}$;

2. Each Π_i is a subset of the attributes in $\mathcal{A} \setminus \{X_i\}$. We say that Π_i is the parent set of X_i in $\mathcal{N}$;

3. For any $1 \leq i < j \leq d$, we have $X_j \notin \Pi_i$, i.e., there is no edge from X_j to X_i in $\mathcal{N}$. This ensures that the network is acyclic, namely, it is a DAG.

We define the *degree* of $\mathcal{N}$ as the maximum size of any parent set Π_i in $\mathcal{N}$. For example, Table 3.1 shows attribute-parent pairs in the Bayesian network $\mathcal{N}_1$ in Figure 3.1; $\mathcal{N}_1$'s degree equals 3, since the parent set of any attribute in $\mathcal{N}_1$ has a size at most three.

Let $\Pr[\mathcal{A}]$ denote the full distribution of tuples in database D. The d attribute-parent pairs in $\mathcal{N}$ essentially define a way to approximate $\Pr[\mathcal{A}]$ with d conditional distributions $\Pr[X_1 \mid \Pi_1], \Pr[X_2 \mid \Pi_2], \ldots, \Pr[X_d \mid \Pi_d]$. In particular, under the assumption that any X_i and any $X_j \notin \Pi_i$ are conditionally independent given Π_i, we have

$$\begin{aligned}\Pr[\mathcal{A}] &= \Pr[X_1, X_2, \ldots, X_d] \\ &= \Pr[X_1] \cdot \Pr[X_2 \mid X_1] \cdot \Pr[X_3 \mid X_1, X_2] \ldots \Pr[X_d \mid X_1, \ldots X_{d-1}] \\ &= \prod_{i=1}^{d} \Pr[X_i \mid \Pi_i]. \end{aligned} \tag{3.1}$$

Table 3.2: Table of notations

notation	description
D	a sensitive dataset to be published
n	the number of tuples in D
$\mathcal{A}$	the set of attributes in D
d	the number of attributes in $\mathcal{A}$
$\mathcal{N}$	a Bayesian network over $\mathcal{A}$
$\Pr[\mathcal{A}]$	the distribution of tuples in D
$\Pr_{\mathcal{N}}[\mathcal{A}]$	an approximation of $\Pr[\mathcal{A}]$ defined by $\mathcal{N}$
$\mathrm{dom}(X)$	the domain of random variable X

Let $\Pr_{\mathcal{N}}[\mathcal{A}] = \prod_{i=1}^{d} \Pr[X_i \mid \Pi_i]$ be the above approximation of $\Pr[\mathcal{A}]$ defined by $\mathcal{N}$. Intuitively, if $\mathcal{N}$ accurately captures the conditional independence among the attributes in $\mathcal{A}$, then $\Pr_{\mathcal{N}}[\mathcal{A}]$ would be a good approximation of $\Pr[\mathcal{A}]$. In addition, if the degree of $\mathcal{N}$ is small, then the computation of $\Pr_{\mathcal{N}}[\mathcal{A}]$ is relatively simple as it requires only d low-dimensional distributions $\Pr[X_1 \mid \Pi_1], \Pr[X_2 \mid \Pi_2], \ldots, \Pr[X_d \mid \Pi_d]$. Low-degree Bayesian networks are the core of our solution to release multi-dimensional data. Table 3.2 shows notations that will be frequently used in this chapter.

3.2 Solution Overview

This section presents an overview of PrivBayes, our solution for releasing a multi-dimensional dataset D in an ε-differentially private manner. PrivBayes runs in three phases:

1. Construct a k-degree Bayesian network $\mathcal{N}$ over the attributes in D, using an $(\varepsilon/2)$-differentially private method. (k is a small value that can be chosen automatically by PrivBayes.)

2. Use an $(\varepsilon/2)$-differentially private algorithm to generate a set of *conditional distributions* of D, such that for each attribute-parent pair (X_i, Π_i) in $\mathcal{N}$, we have a noisy version of the conditional distribution $\Pr[X_i \mid \Pi_i]$. (We denote this noisy distribution as $\Pr^*[X_i \mid \Pi_i]$.)

3. Use the Bayesian network $\mathcal{N}$ (constructed in the first phase) and the d noisy conditional distributions (constructed in the second phase) to derive an approximate distribution of the tuples in D, and then sample tuples from the approximate distribution to generate a synthetic dataset D^*.

In short, PrivBayes utilizes a low-degree Bayesian network $\mathcal{N}$ to generate a synthetic dataset D^* that approximates the multi-dimensional input data D. The construction of

Algorithm 3.1: NoisyConditionals $(D, \mathcal{N}, k)$: returns $\mathcal{P}^*$

1 initialize $\mathcal{P}^* = \emptyset$;
2 **for** $i = k + 1$ **to** d **do**
3 materialize the joint distribution $\Pr[X_i, \Pi_i]$;
4 generate differentially private $\Pr^*[X_i, \Pi_i]$ by adding noise Lap $\left(\frac{4 \cdot (d-k)}{n \cdot \varepsilon}\right)$;
5 set negative values in $\Pr^*[X_i, \Pi_i]$ to 0 and normalize;
6 derive $\Pr^*[X_i \mid \Pi_i]$ from $\Pr^*[X_i, \Pi_i]$; add it to $\mathcal{P}^*$;
7 **for** $i = 1$ **to** k **do**
8 derive $\Pr^*[X_i \mid \Pi_i]$ from $\Pr^*[X_{k+1}, \Pi_{k+1}]$; add it to $\mathcal{P}^*$;
9 **return** $\mathcal{P}^*$;

$\mathcal{N}$ is highly non-trivial, as it requires carefully selecting attribute-parent pairs and the value of k to derive a close approximation of D without violating differential privacy. By contrast, the second and third phases of PrivBayes are relatively straightforward. In the following, we will clarify the details of these phases, and prove the privacy guarantee of PrivBayes; the algorithm for PrivBayes's first phase will be elaborated in Section 3.3.

Generation of Noisy Conditional Distributions. Suppose that we are given a k-degree Bayesian network $\mathcal{N}$. To construct the approximate distribution $\Pr_{\mathcal{N}}[\mathcal{A}]$, we need d conditional distributions $\Pr[X_i \mid \Pi_i]$ $(i \in [1, d])$, as shown in Equation (3.1). Algorithm 3.1 illustrates how the distributions specified by our algorithm can be derived in a differentially private manner. In particular, for any $i \in [k + 1, d]$, the algorithm first materializes the joint distribution $\Pr[X_i, \Pi_i]$ (Line 3), and then injects Laplace noise into $\Pr[X_i, \Pi_i]$ to obtain a noisy distribution $\Pr^*[X_i, \Pi_i]$ (Lines 4-5). To enforce the fact that these are probability distributions, all negative numbers in $\Pr^*[X_i, \Pi_i]$ are set to zero, then all values are normalized to maintain a total probability mass of 1 (Line 5).[1] After that, based on $\Pr^*[X_i, \Pi_i]$, the algorithm derives a noisy version of the conditional distribution $\Pr[X_i \mid \Pi_i]$, denoted as $\Pr^*[X_i \mid \Pi_i]$ (Line 6). The scale of the Laplace noise added to $\Pr[X_i, \Pi_i]$ is set to $4(d-k)/n\varepsilon$, which ensures that the generation of $\Pr^*[X_i, \Pi_i]$ satisfies $(\varepsilon/2(d-k))$-differential privacy, since $\Pr[X_i, \Pi_i]$ has sensitivity $2/n$. Meanwhile, the derivation of $\Pr^*[X_i \mid \Pi_i]$ from $\Pr^*[X_i, \Pi_i]$ does not incur any privacy cost, as it only relies on $\Pr^*[X_i, \Pi_i]$ instead of the input data D.

Overall, Lines 2-6 of Algorithm 3.1 construct $(d - k)$ noisy conditional distributions $\Pr^*[X_i \mid \Pi_i]$ $(i \in [k+1, d])$, and they satisfy $(\varepsilon/2)$-differential privacy, since each $\Pr^*[X_i \mid$

[1] More generally, we could apply additional post-processing of distributions, in the spirit of [Barak et al., 2007; Ding et al., 2011; Hay et al., 2010], to reflect the fact that lower degree distributions should be consistent. Interested readers are referred to Section 4.3 of [Chen et al., 2015] for the current state-of-the-art.

$\Pi_i]$ is $(\varepsilon/2(d-k))$-differentially private. This is due to the composability property of differential privacy [Dwork, 2006]. In particular, composability indicates that when a set of m algorithms satisfy differential privacy with parameters $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m$, respectively, the set of algorithms as a whole satisfies $(\sum_i \varepsilon_i)$-differential privacy.

After $\Pr^*[X_{k+1} \mid \Pi_{k+1}], \ldots, \Pr^*[X_d \mid \Pi_d]$ are constructed, Algorithm 3.1 proceeds to generate $\Pr^*[X_i \mid \Pi_i]$ $(i \in [1,k])$. This generation, however, does not require any additional information from the input data D. Instead, we derive $\Pr^*[X_i \mid \Pi_i]$ $(i \in [1,k])$ directly from $\Pr^*[X_{k+1}, \Pi_{k+1}]$, which has been computed in Lines 2-6 of Algorithm 3.1. Such derivation is feasible, since our algorithm for constructing the Bayesian network $\mathcal{N}$ (to be clarified in Section 3.3) ensures that $X_i \in \Pi_{k+1}$ and $\Pi_i \subset \Pi_{k+1}$ for any $i \in [1,k]$. Since each $\Pr^*[X_i \mid \Pi_i]$ $(i \in [1,k])$ is derived from $\Pr^*[X_{k+1}, \Pi_{k+1}]$ without inspecting D, the construction of $\Pr^*[X_i \mid \Pi_i]$ does not incur any privacy overhead. Therefore, Algorithm 3.1 as a whole is $(\varepsilon/2)$-differentially private. Example 3.1 illustrates Algorithm 3.1.

Example 3.1 *Suppose that we are given a 2-degree Bayesian network $\mathcal{N}$ over a set of four attributes $\{A, B, C, D\}$, with 4 attribute-parent pairs: $(A, \emptyset), (B, \{A\}), (C, \{A, B\})$, and $(D, \{A, C\})$. Given $\mathcal{N}$, Algorithm 3.1 constructs two noisy joint distributions $\Pr^*[A, B, C]$ and $\Pr^*[A, C, D]$. Based on $\Pr^*[A, C, D]$, Algorithm 3.1 derives a noisy conditional distribution $\Pr^*[D \mid A, C]$. In addition, the algorithm uses $\Pr^*[A, B, C]$ to derive three other conditional distributions $\Pr^*[A]$, $\Pr^*[B \mid A]$, and $\Pr^*[C \mid A, B]$. Given these four conditional distributions, the input tuple distribution is approximated as*

$$\Pr^*_{\mathcal{N}}[A, B, C, D] = \Pr^*[A] \cdot \Pr^*[B \mid A] \cdot \Pr^*[C \mid A, B] \cdot \Pr^*[D \mid A, C].$$

Generation of Synthetic Data. Even with the simple closed-form expression in Equation 3.1, it is still time and space consuming to directly sample from $\Pr^*_{\mathcal{N}}[\mathcal{A}]$ by computing the probability for each element in the domain of $\mathcal{A}$. Fortunately, the Bayesian network $\mathcal{N}$ provides a means to perform sampling efficiently without materializing $\Pr^*_{\mathcal{N}}[\mathcal{A}]$. As shown in Equation (3.1), we can sample each X_i from the conditional distribution $\Pr^*[X_i \mid \Pi_i]$ independently, without considering any attribute not in $\Pi_i \cup \{X_i\}$. Furthermore, the properties of $\mathcal{N}$ (discussed in Section 3.1) ensure that $X_j \notin \Pi_i$ for any $j > i$. Therefore, if we sample X_i $(i \in [1,d])$ in increasing order of i, then by the time X_j $(j \in [2,d])$ is to be sampled, we must have sampled all attributes in Π_j, i.e., we will be able to sample X_j from $\Pr^*[X_j \mid \Pi_j]$ given the previously sampled attributes. That is to say, the sampling of X_j does not require the full distribution $\Pr^*_{\mathcal{N}}[\mathcal{A}]$.

With the above sampling approach, we can generate an arbitrary number of tuples from $\Pr^*_{\mathcal{N}}[\mathcal{A}]$ to construct a synthetic database D^*. In this chapter, we consider the size of D^* is set to n, i.e., the same as the number of tuples in the input data D.

Privacy Guarantee. The correctness of PrivBayes directly follows the composability property of differential privacy [Dwork, 2006]. In particular, the first and second phases of PrivBayes require direct access to the input database, and each of them consumes $\varepsilon/2$ privacy budget. No access to the original database is invoked during the third (sampling) phase. The results of first two steps, i.e., the Bayesian network $\mathcal{N}$ and the set of noisy conditional distributions, are sufficient to generate the synthetic database D^*. Therefore, we have the following theorem.

Theorem 3.1 PrivBayes *satisfies ε-differential privacy.*

3.3 Private Bayesian Networks

This section presents our solution for constructing differentially private Bayesian networks. We will first introduce a non-private algorithm for Bayesian network construction (in Section 3.3.1), and then explain how the algorithm can be converted into a differentially private solution (in Sections 3.3.2 and 3.3.3).

3.3.1 Non-Private Methods

Suppose that we aim to construct a k-degree Bayesian network $\mathcal{N}$ on a dataset D containing a set $\mathcal{A}$ of attributes. Ideally, $\mathcal{N}$ should provide an accurate approximation of the tuple distribution in D, i.e., $\Pr_{\mathcal{N}}[\mathcal{A}]$ should be close to $\Pr[\mathcal{A}]$. A natural question is under what condition will $\Pr_{\mathcal{N}}[\mathcal{A}]$ closely approximate $\Pr[\mathcal{A}]$? We make use of standard notions from information theory to measure this. The *entropy* of a random variable X over its domain $\mathrm{dom}(X)$ is denoted by $H(X) = -\sum_{x\in\mathrm{dom}(X)} \Pr[X = x] \log \Pr[X = x]$,[2] and $I(\cdot, \cdot)$ denotes the *mutual information* between two variables as:

$$I(X, \Pi) = \sum_{x\in\mathrm{dom}(X)} \sum_{\pi\in\mathrm{dom}(\Pi)} \Pr[X = x, \Pi = \pi] \log \frac{\Pr[X = x, \Pi = \pi]}{\Pr[X = x] \Pr[\Pi = \pi]}, \tag{3.2}$$

where $\Pr[X, \Pi]$ is the joint distribution of X and Π, and $\Pr[X]$ and $\Pr[\Pi]$ are the marginal distributions of X and Π respectively. The *KL-divergence* [Cybakov, 2009a] of $\Pr_{\mathcal{N}}[\mathcal{A}]$ from $\Pr[\mathcal{A}]$ measures the difference between the two probability distributions, and is defined by:

$$KL(\Pr[\mathcal{A}], \Pr_{\mathcal{N}}[\mathcal{A}]) = -\sum_{i=1}^{d} I(X_i, \Pi_i) + \sum_{i=1}^{d} H(X_i) - H(\mathcal{A}). \tag{3.3}$$

[2] All logarithms used in this chapter are to the base 2.

Algorithm 3.2: GreedyBayes (D, k): returns $\mathcal{N}$

1 initialize $\mathcal{N} = \emptyset$ and $V = \emptyset$;
2 randomly select an attribute X_1 from $\mathcal{A}$; add $(X_1, \emptyset)$ to $\mathcal{N}$; add X_1 to V;
3 **for** $i = 2$ **to** d **do**
4 initialize $\Omega = \emptyset$;
5 for each $X \in \mathcal{A} \backslash V$ and each $\Pi \in \binom{V}{k}$, add (X, Π) to Ω;
6 select a pair (X_i, Π_i) from Ω with the maximal mutual information $I(X_i, \Pi_i)$;
7 add (X_i, Π_i) to $\mathcal{N}$; add X_i to V;
8 **return** $\mathcal{N}$;

We seek a Bayesian network representation so that the KL-divergence between the original and the approximate distribution is small. In (3.3), the term $\sum_{i=1}^{d} H(X_i) - H(\mathcal{A})$ is solely decided by $\Pr[\mathcal{A}]$, which is fixed once the input database D is given. Hence, the KL-divergence of $\Pr_{\mathcal{N}}[\mathcal{A}]$ from $\Pr[\mathcal{A}]$ is small (in which case they closely approximate each other), if and only if $\sum_{i=1}^{d} I(X_i, \Pi_i)$ is maximized. Therefore, the construction of $\mathcal{N}$ can be modeled as an optimization problem, where we aim to choose a parent set Π_i for each attribute X_i in D to maximize $\sum_{i=1}^{d} I(X_i, \Pi_i)$.

For the case when $k = 1$, Chow and Liu show that greedily picking the next edge based on the maximum mutual information is optimal, leading to the celebrated notion of Chow-Liu trees [Chow and Liu, 1968]. However, as shown in [Chickering et al., 2004], this optimization problem is NP-hard when $k > 1$. For this reason, heuristic algorithms (e.g., hill-climbing, genetic algorithms, and simulated annealing) are often employed in practice [Margaritis, 2003]. In the context of differential privacy, however, a different calculus applies: these methods incur a high cost in terms of sensitivity and so incur a large amount of noise. That is, these algorithms make many queries to the data, so that making them differentially private entails large perturbations which lead to poor overall accuracy. Therefore, we seek a new method that will imply less noise, and so give a better overall approximation when the noise is added. Thus we propose a greedy algorithm that makes fewer probes to the data, by extending the Chow-Liu approach to higher degrees, described in Algorithm 3.2.

In the beginning of the algorithm (Line 1), we initialize the Bayesian network $\mathcal{N}$ to an empty list of attribute-parent pairs. Let V be a set that contains all attributes whose parent sets have been fixed in the partial construction of $\mathcal{N}$. As a next step, the algorithm randomly selects an attribute (denoted as X_1) from $\mathcal{A}$, and sets its parent set Π_1 to $\emptyset$ (Line 2). The rest of the algorithm consists of $d - 1$ iterations (Lines 3-7), in each of which we greedily add into $\mathcal{N}$ an attribute-parent pair with a large mutual information. Specifically, the attribute-parent pair in each iteration is selected from a candidate set Ω that contains every attribute-parent pair (X, Π) satisfying two requirements:

1. $|\Pi| \leq k$, which ensures that $\mathcal{N}$ is a k-degree Bayesian network. This is ensured by choosing Π only from $\binom{V}{k}$, where $\binom{V}{k}$ denotes the set of all subsets of V with size $\min(k, |V|)$ (Lines 5-6).

2. $\mathcal{N}$ contains no edge from X_i to X_j for any $j < i$, which guarantees that $\mathcal{N}$ is a DAG. We ensure this condition by requiring that in the beginning of any iteration, V only contains the attributes whose parent sets have been decided in the previous iterations (Line 7). In other words, the parent set of X_i can only be a subset of $\{X_1, X_2, \ldots, X_{i-1}\}$, as a consequence of which $\mathcal{N}$ cannot contain any edge from X_i to X_j for any $j < i$.

Once the parent set of each attribute is decided, the algorithm terminates and returns the Bayesian network $\mathcal{N}$ (Line 8). The number of pairs considered in iteration i is $(d-i)\binom{i}{k}$, so summing over all iterations the cost is bounded by $d\sum_{i=1}^{d}\binom{i}{k} = d\binom{d+1}{k+1}$. This determines the asymptotic cost of the procedure. Note that when $k = 1$, the above algorithm is equivalent to Chow and Liu's method [Chow and Liu, 1968] for constructing optimal 1-degree Bayesian networks.

3.3.2 A First-Cut Solution

Observe that in Algorithm 3.2, there is only one place where we interact directly with the input dataset D, namely, the greedy selection of an attribute-parent pair (X_i, Π_i) in each iteration of the algorithm (Line 6). Therefore, if we are to make Algorithm 3.2 differentially private, we only need to replace Line 6 of Algorithm 3.2 with a procedure that selects (X_i, Π_i) from Ω in a private manner. Such a procedure can be implemented with the exponential mechanism outlined in Section 2.2, using the mutual information function I as the quality function. Specifically, we first inspect each attribute-parent pair $(X, \Pi) \in \Omega$, and calculate the mutual information $I(X, \Pi)$ between X and Π; After that, we sample an attribute-parent pair from Ω, such that the sampling probability of any pair (X, Π) is proportional to $\exp(I(X, \Pi)/2\Delta)$, where Δ is a scaling factor.

The value of Δ is set as follows. As mentioned in Section 3.2, PrivBayes requires that the construction of the Bayesian network $\mathcal{N}$ should satisfy $(\varepsilon/2)$-differential privacy. Accordingly, we set $\Delta = 2(d-1)S(I)/\varepsilon$, where $S(I)$ denotes the sensitivity of the mutual information function I (see Equation (2.4)). This ensures that each invocation of the exponential mechanism satisfies $(\varepsilon/2(d-1))$-differential privacy. Given the composability property of differential privacy [Dwork, 2006] and the fact that we only invoke the exponential mechanism $d-1$ times during the construction of $\mathcal{N}$, it can be verified that the overall process of constructing $\mathcal{N}$ is $(\varepsilon/2)$-differentially private.

Last, we calculate the sensitivity, $S(I)$.

Lemma 3.1

$$S(I(X,\Pi)) = \begin{cases} \frac{1}{n}\log n + \frac{n-1}{n}\log\frac{n}{n-1}, & \textit{if } X \textit{ or } \Pi \textit{ is binary;} \\ \frac{2}{n}\log\frac{n+1}{2} + \frac{n-1}{n}\log\frac{n+1}{n-1}, & \textit{otherwise,} \end{cases}$$

where n is the number of tuples in D.

This can be shown by considering the maximum change in mutual information based on its definition (3.2), as the various probabilities are changed by the alteration of one tuple. We defer the full proof to Appendix B for brevity, but the maximum difference in mutual information between binary variables is achieved by this example:

$X\backslash\Pi$	0	1
0	0	0
1	0	1

$X\backslash\Pi$	0	1
0	$\frac{1}{n}$	0
1	0	$\frac{n-1}{n}$

The mutual information of the left distribution is 0, and that of the right one is $\frac{1}{n}\log n + \frac{n-1}{n}\log\frac{n}{n-1}$.

3.3.3 An Improved Solution

The method in Section 3.3.2 is simple and intuitive, but may not achieve the best results: Observe that $S(I) > \log n/n$; this can be large compared to the range of I. For example, range$(I) = 1$ for binary distributions. As a consequence, the scaling factor $\Delta = 2(d-1)S(I)/\varepsilon$ tends to be large, and so the exponential mechanism is still quite likely to sample (from Ω) an attribute-parent pair with a small mutual information. In that case, the Bayesian network $\mathcal{N}$ constructed using the exponential mechanism will offer a weak approximation of $\Pr[\mathcal{A}]$, resulting in a low-quality output from PrivBayes. To improve over this solution, we propose to avoid using I as the quality function in the exponential mechanism. Instead, we define a novel function F that maps each attribute-parent pair $(X,\Pi) \in \Omega$ to a score, such that

1. F's sensitivity is small (with respect to the range of F).
2. If $F(X,\Pi)$ is large, then $I(X,\Pi)$ tends to be large.

The rationale is that since $S(F)$ is small with respect to range(F), the scaling factor $\Delta = 2(d-1)S(F)/\varepsilon$ will also be small, and hence, the exponential mechanism has a high probability to select an attribute-parent pair (X,Π) with a large $F(X,\Pi)$. In turn, such an attribute-parent pair tends to have a large mutual information between X and Π, which helps improve the quality of the Bayesian network $\mathcal{N}$.

In what follows, we will clarify our construction of F. To achieve property 2 above, we set F to its maximum value (i.e., 0) when I is greatest. To achieve property 1, we make

$F(X, \Pi)$ decrease linearly in proportion to the L_1 distance from $\Pr[X, \Pi]$ to a distribution that maximizes F, since linear functions ensure that the sensitivity is controlled: the function does not change sharply anywhere in its domain. We first introduce the concept of *maximum joint distribution*, which will be used to define the peaks of F, and then characterize such distributions:

Definition 3.1 (Maximum Joint Distribution) *Given an attribute-parent pair (X, Π), a maximum joint distribution $\Pr^\diamond[X, \Pi]$ for X and Π is one that maximizes the mutual information between X and Π.*

Lemma 3.2 *Assume that $|\mathrm{dom}(X)| \leq |\mathrm{dom}(\Pi)|$. A distribution $\Pr^\diamond[X, \Pi]$ is a maximum joint distribution if and only if*

1. *$\Pr^\diamond[X = x] = 1/|\mathrm{dom}(X)|$, for any $x \in \mathrm{dom}(X)$;*
2. *For any $\pi \in \mathrm{dom}(\Pi)$, there is at most one $x \in \mathrm{dom}(X)$ with $\Pr^\diamond[X = x, \Pi = \pi] > 0$.*

Proofs in this section are deferred to Appendix B. We illustrate Definition 3.1 and Lemma 3.2 with an example:

Example 3.2 *Consider a binary variable X with $\mathrm{dom}(X) = \{0, 1\}$ and a variable Π with $\mathrm{dom}(\Pi) = \{a, b, c\}$. Consider two joint distributions between X and Π as follows:*

$X \backslash \Pi$	a	b	c
0	.5	0	0
1	0	.5	0

$X \backslash \Pi$	a	b	c
0	0	.2	.3
1	.5	0	0

By Lemma 3.2, both of the above distributions are maximum joint distributions, with $I(X, \Pi) = 1$.

Let (X, Π) be an attribute-parent pair, and $\mathcal{P}^\diamond[X, \Pi]$ be the set of all maximum joint distributions for X and Π. Our quality function F (for evaluating the quality of (X, Π)) is defined as

$$F(X, \Pi) = -\frac{1}{2} \min_{\Pr^\diamond \in \mathcal{P}^\diamond} \left\| \Pr[X, \Pi] - \Pr^\diamond[X, \Pi] \right\|_1. \tag{3.4}$$

If $F(X, \Pi)$ is large, then $\Pr[X, \Pi]$ must have a small L_1 distance to one of the maximum joint distributions in $\mathcal{P}^\diamond[X, \Pi]$, and vice-versa. In turn, if $\Pr[X, \Pi]$ is close to a maximum joint distribution in $\mathcal{P}^\diamond[X, \Pi]$, then intuitively, $\Pr[X, \Pi]$ is likely to give a large mutual information between X and Π. In other words, the value of $F(X, \Pi)$ tends to be positively correlated with $I(X, \Pi)$. This explains why F could be a good quality function to replace I. In addition, F has a much smaller sensitivity than I, as shown in the following theorem:

Theorem 3.2 $S(F) = 1/n$.

This follows immediately from considering the L_1 distance between neighboring distributions. Observe that $S(F) < S(I)/\log n$, where n is the number of tuples in the input data. Meanwhile, the ranges of F and I are comparable; for example, range$(I) = 1$ and range$(F) = 0.5$ for binary domains. Therefore, when n is large (as is often the case), the sensitivity-to-range ratio of F is significantly smaller than that of I, which makes F a favorable quality function over I for selecting attribute-parent pairs in the Bayesian network $\mathcal{N}$.

3.3.4 Computation of F

While (3.4) defines the function F, it still remains unclear how we can calculate $F(X, \Pi)$ given $\Pr[X, \Pi]$. In this subsection, we use dynamic programming to solve the problem for the case when all attributes in $\Pi \cup \{X\}$ have binary domains; we address the case of non-binary domains in Section 3.4.

Let (X, Π) be an attribute-parent pair where $|\Pi| = k$. Then, the joint distribution $\Pr[X, \Pi]$ can be represented by a 2×2^k matrix where the sum of all elements is 1. For example, Table 3.3(a) illustrates a joint distribution $\Pr[X, \Pi]$ with $|\Pi| = 2$. To compute $F(X, \Pi)$, we need to identify the minimum L_1 distance between $\Pr[X, \Pi]$ and a maximum joint distribution $\Pr^\diamond[X, \Pi] \in \mathcal{P}^\diamond[X, \Pi]$. Table 3.3(b) illustrates one such maximum joint distribution, whose L_1 distance to the distribution in Table 3.3(a) equals 0.4. To derive the minimum L_1 distance, a naive approach is to enumerate all maximum joint distributions in $\mathcal{P}^\diamond[X, \Pi]$; nevertheless, as $\mathcal{P}^\diamond[X, \Pi]$ may contain an infinite number of maximum joint distributions, a brute-force enumeration of $\mathcal{P}^\diamond$ is infeasible. To address this issue, we will first introduce an exponential-time algorithm for computing $F(X, \Pi)$, which will serve as the basis of our dynamic programming solution.

The basic idea of our exponential-time algorithm is to (i) partition the distributions in $\mathcal{P}^\diamond$ into a finite number of equivalence classes, and then (ii) compute $F(X, \Pi)$ by processing each equivalence class individually. By Lemma 3.2, any maximum joint distribution $\Pr^\diamond[X, \Pi]$ has the following property: for any $\pi \in \text{dom}(\Pi)$, either $\Pr^\diamond[X = 0, \Pi = \pi] = 0$ or $\Pr^\diamond[X = 1, \Pi = \pi] = 0$. In other words, for each column in the matrix representation of $\Pr^\diamond[X, \Pi]$ (where a column corresponds to a value in dom(Π)), there should be at most one non-zero entry. For example, the gray cells in Table 3.3(b) indicate the positions of non-zeros in the given maximum joint distribution.

For any two distributions in $\mathcal{P}^\diamond$, we say that they are *equivalent* if their positions of the non-zero entries are the same. Suppose that we divide the distributions in $\mathcal{P}^\diamond$ into equivalence classes, each of which contains a maximal subset of equivalent distributions. Then, totally there are $O(3^{2^k})$ equivalent classes. It turns out that we can easily calculate the minimum L_1 distance from $\Pr[X, \Pi]$ to each equivalence class.

Table 3.3: An example of joint distributions

$X\backslash\Pi$	00	01	10	11
0	.6	0	0	0
1	.1	.1	.1	.1

(a) input joint distribution

$X\backslash\Pi$	00	01	10	11
0	.5	0	0	0
1	0	.3	.1	.1

(b) maximum joint distribution

To explain, consider a particular equivalence class E. Let Z^- be the set of pairs (x, π), such that $\Pr^\diamond[X = x, \Pi = \pi] = 0$ for any $\Pr^\diamond[X, \Pi] \in E$. That is, Z^- captures the positions of all zero entries in the matrix representation of $\Pr^\diamond[X = x, \Pi = \pi]$. Similarly, we define the sets of non-zero entries in row $X = 0$ and $X = 1$ as

$$Z_0^+ = \{(0, \pi) \mid \Pr^\diamond[X = 0, \Pi = \pi] > 0\} \text{ and } Z_1^+ = \{(1, \pi) \mid \Pr^\diamond[X = 1, \Pi = \pi] > 0\}.$$

For convenience, we also abuse notation and define

$$\begin{aligned}
\Pr[Z^-] &= \sum\nolimits_{(x,\pi)\in Z^-} \Pr[X = x, \Pi = \pi], \\
\Pr[Z_0^+] &= \sum\nolimits_{(x,\pi)\in Z_0^+} \Pr[X = x, \Pi = \pi], \\
\Pr[Z_1^+] &= \sum\nolimits_{(x,\pi)\in Z_1^+} \Pr[X = x, \Pi = \pi].
\end{aligned}$$

By Lemma 3.2, we have $\Pr^\diamond[Z^-] = 0$, $\Pr^\diamond[Z_0^+] = 1/2$, and $\Pr^\diamond[Z_1^+] = 1/2$ for any $\Pr^\diamond[X, \Pi] \in E$. Then, for any $\Pr[X, \Pi]$, its L_1 distance to a distribution $\Pr^\diamond[X, \Pi] \in E$ is bounded by:

$$\left\| \Pr[X, \Pi] - \Pr^\diamond[X, \Pi] \right\|_1 \geq \Pr[Z^-] + \left| \Pr[Z_0^+] - \frac{1}{2} \right| + \left| \Pr[Z_1^+] - \frac{1}{2} \right|.$$

Let $(x)_+$ denote $\max(0, x)$. Given that $\Pr[Z^-]+\Pr[Z_0^+]+\Pr[Z_1^+] = 1$, the above inequality can be simplified to

$$\left\| \Pr[X, \Pi] - \Pr^\diamond[X, \Pi] \right\|_1 \geq 2 \cdot \left[\left(\tfrac{1}{2} - \Pr[Z_0^+] \right)_+ + \left(\tfrac{1}{2} - \Pr[Z_1^+] \right)_+ \right]. \tag{3.5}$$

Furthermore, there always exists a $\Pr^\diamond[X, \Pi] \in E$ that makes the equality hold. In other words, once the positions of the non-zero entries in $\Pr^\diamond[X, \Pi]$ are fixed, we can use (3.5) to derive the minimum L_1 distance from any $\Pr[X, \Pi]$ to E, with a linear scan of the entries in the matrix representation of $\Pr[X, \Pi]$. By enumerating all $O(3^{2^k})$ equivalence classes of $\mathcal{P}^\diamond$, we can then derive $F(X, \Pi)$.

The above procedure for calculating F is impractical when $k \geq 4$, as the exhaustive search over all possible equivalence classes of $\mathcal{P}^\diamond$ is prohibitive. To tackle this problem, we propose a dynamic-programming-based optimization that reduces computation costs by taking advantage of the fact that the distributions are induced by n items.

Based on (3.5), our target is to find a combination of Z_0^+ and Z_1^+ (which therefore determine Z^-) that minimizes

$$\left(\frac{1}{2} - \Pr[Z_0^+]\right)_+ + \left(\frac{1}{2} - \Pr[Z_1^+]\right)_+.$$

We define the probability mass associated with Z_0^+ and Z_1^+ as K_0 and K_1 respectively. Initially, $K_0 = K_1 = 0$. For each $\pi \in \mathrm{dom}(\Pi)$, we can either increase K_0 by $\Pr[X = 0, \Pi = \pi]$ (by assigning $(0, \pi)$ to Z_0^+) or increase K_1 by $\Pr[X = 1, \Pi = \pi]$ (by assigning $(1, \pi)$ to Z_1^+). We index $\pi \in \mathrm{dom}(\Pi)$ as $\pi_1, \pi_2, \ldots, \pi_{2^k}$. We use $C(i, a, b)$ to indicate if $K_0 = a/n$ and $K_1 = b/n$ is reachable by using the first i π's, i.e., $\pi_1, \pi_2, \ldots, \pi_i$. It can be verified that (i) $C(i, a, b) =$ true if $i = a = b = 0$, (ii) $C(i, a, b) =$ false if $i < 0$ or $a < 0$ or $b < 0$, and (iii) otherwise,

$$C(i, a, b) = C(i-1, a-n\Pr[X = 0, \Pi = \pi_i], b) \vee C(i-1, a, b-n\Pr[X = 1, \Pi = \pi_i]). \quad (3.6)$$

Given an input dataset D with n tuple, each cell in $\Pr[X, \Pi]$ must be a multiple of $1/n$. Thus, we only consider the case when a and b are integers in the range $[0, n]$. Thus, the total number of states $C(i, a, b)$ is $n^2 2^k$. A direct traversal of all states takes $O(n^2 2^k)$ time. To reduce this time complexity, we introduce the following concept:

Definition 3.2 (Dominated State) *A state $C(i, a_1, b_1)$ is* dominated *by $C(i, a_2, b_2)$ if and only if $a_1 \leq a_2$ and $b_1 \leq b_2$.*

Note that a dominated state is always inferior to some other states, and hence, it can be ignored without affecting the correctness of final result. Consequently, we maintain the set of at most n non-dominated reachable states for each $i \in [1, 2^k]$. The function F can be calculated by

$$F(X, \Pi) = -\min_{C(2^k, a, b) = \text{true}} \left(\frac{1}{2} - \frac{a}{n}\right)_+ + \left(\frac{1}{2} - \frac{b}{n}\right)_+.$$

As such, the total number of states that need to be traversed is $n2^k$, and thus the complexity of the algorithm is reduced to $O(n2^k)$. Note that k is small in practice, since we only need to consider low-degree Bayesian networks.

3.3.5 Choice of k and θ-usefulness.

We have discussed how to build a k-degree Bayesian network under differential privacy, where k is considered as a given input to the algorithm. However, k is usually unknown in real applications and should be chosen carefully. The choice of k is non-trivial for PrivBayes. Intuitively, a Bayesian network with a larger k keeps more information

from the full dimensional distribution $\Pr[\mathcal{A}]$, e.g., a $(d-1)$-degree Bayesian network approximates $\Pr[\mathcal{A}]$ perfectly without having any information loss. On the other hand, the downside of using large k is that it forces PrivBayes to anonymize a set of higher dimensional marginal distributions in the second phase, which are very vulnerable to noise due to their domains of large size. These noisy distributions are less useful after anonymization especially when the privacy budget ε is small, leading to a synthetic database full of random perturbation. With very small values of ε, the best choice may be to pick $k = 0$, i.e. to model all attributes as independent. Hence, the choice of k should balance the informativeness of a Bayesian network and the robustness of marginal distributions. This balancing act is affected by three parameters: the total privacy budget ε, the total number of tuples in database n, and usefulness of each noisy marginal distribution in the second phase θ. We quantify this in the following definition.

Definition 3.3 (θ-usefulness) *A noisy distribution is θ-useful if the ratio of average probability to average scale of noise in its cells is no less than θ.*

Lemma 3.3 *The noisy distributions in Algorithm 3.1 are* $\left(\frac{n \cdot \varepsilon}{(d-k) \cdot 2^{k+3}}\right)$*-useful.*

Proof: In Algorithm 3.1, each marginal distribution is $(k+1)$-dimensional with a domain size 2^{k+1}. Therefore, the average probability in each cell is $1/2^{k+1}$.

For the scale of noise, we have $d-k$ marginal distributions to be anonymized and each of them consumes privacy budget $\varepsilon/2(d-k)$. The sensitivity of each marginal distribution is $2/n$. According to the Laplace mechanism, the Laplace noise N injected to each cell is drawn from distribution $\mathrm{Lap}\left(4(d-k)/n\varepsilon\right)$ where the average scale of noise is $\mathrm{E}(|\eta|) = 4(d-k)/n\varepsilon$. □

The notion of θ-usefulness provides a more intuitive way to choose k automatically without closely studying the specific instance of the input database. Generally speaking, we believe a 0.5-useful noisy distribution is not good because the scale of noise is twice as that of information, while a 5-useful one is more reliable due to its large information to noise ratio. In practice, we set up a threshold θ, then choose the largest positive integer k that guarantees θ-usefulness in distribution learning (note, this is independent of the data, as it depends only on the non-private values ε, θ, n and d). If such a k does not exist, k is set to the minimum value, 0. In the experimental section, we will show that there is a wide range of θ to choose to train a PrivBayes model for good performance.

3.4 Extensions to General Domains

The dynamic programming approach in Section 3.3.4 and the notion of θ-usefulness in Section 3.3.5 both assume that all attributes in the input data D are binary. In this section, we extend our solution to the case when D contains non-binary attributes.

3.4.1 Encodings of Non-binary Attributes

First, we study how to represent non-binary attributes in PrivBayes. We present four different approaches, starting with the application of a standard encoding idea.

Binary encoding. Following the common practice in the literature [Yaroslavtsev et al., 2013], our first approach to handling non-binary attributes is to convert each of them into a set of binary attributes. In particular, for each categorical attribute X whose domain size equals ℓ, we first encode each value in X's domain into a binary representation with $\lceil \log \ell \rceil$ bits; after that, we convert X into $\lceil \log \ell \rceil$ binary attributes $X_1, X_2, \ldots, X_{\lceil \log \ell \rceil}$, such that X_i corresponds to the i-th bit in the binary representation. Meanwhile, for each continuous attribute Y, we first discretize the domain of Y into a fixed number b of equi-width bins[3], and then convert Y into $\lceil \log b \rceil$ binary attributes, using a similar approach to the transformation of X. Figures 3.2 and 3.3 give examples of binary encoding on continuous and categorical attributes, respectively. Figure 3.2 shows how we might encode the attribute "age". First, it is broken into $b = 8$ (for the purpose of illustration) ranges of ten years each. Then the index for each range is represented as three binary bits (i.e. $\log_2 8$), 000_2 for the first, 111_2 for the last. For the categorical attribute in Figure 3.3, any linearization of the attribute values to order them can be used before the binary encoding is applied. After the transformation, D can be encoded to form a new database D_b in the binary domain. Then, we apply PrivBayes on D_b to generate a synthetic dataset D_b^*, then decode it to get D^* in the original domain.

The strength of this approach is that it allows the direct use of θ-usefulness and the advanced score function F. Moreover, the binary decomposition of attributes provides a high level of flexibility in constructing Bayesian networks. For instance, assume that the value of a binary attribute "is retired" is true, if and only if the value of "age" (as in Figure 3.2) is larger than 60. To fully preserve this correlation, PrivBayes with binary encoding requires to use the binary attribute "is retired" and merely the first two bits of "age". The reason is that the first two bits actually discretize the domain of "age" into four bins, i.e., $(0, 20], (20, 40], (40, 60]$ and $(60, 80]$, which is sufficient for the purpose. Data utility is then improved, as the corresponding joint distribution of "is retired" and "age" contains only 8 cells, instead of 16 if all bits are included; thus, it is more robust against noise.

Although the binary encoding provides high flexibility, it comes at the cost of some redundancy. The semantics of the new binary attributes in isolation is not always very clear. For example, the second bit of "age" in Figure 3.2 represents whether a person's age is in $(20, 40] \cup (60, 80]$. This alone does not make too much sense, until the most significant bit is taken into account as well. As for the categorical attribute "workclass"

[3]We use $b = 16$ in experiments, and $b = 8$ in the example in Figure 3.2 for simplicity of presentation.

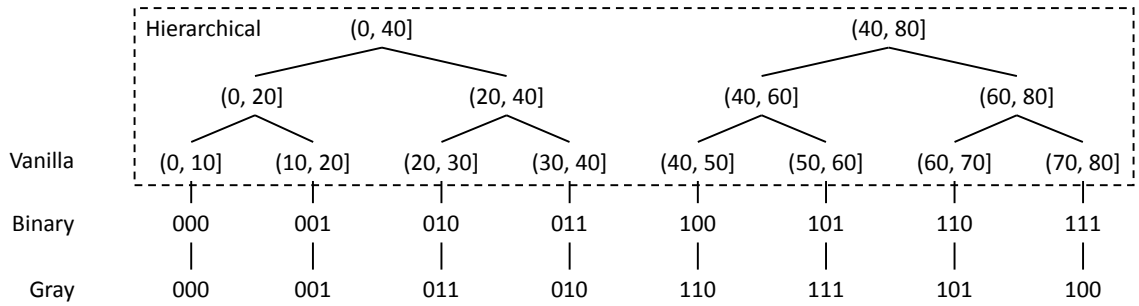


Figure 3.2: Four encodings on a continuous attribute "age"

(in Figure 3.3), each bit alone eventually splits all values in the domain into two subsets. But there is no semantics behind those partitions, unless all three bits (and partitions) are combined together. In the construction of a Bayesian network, however, all these bits are treated as natural attributes of independent interest, leading to the consideration of a large number of redundant attribute-parent pairs with artificial meanings. This would hurt the quality of Bayesian networks, if any redundant attribute-parent pair is selected, which may often be the case when ε is small.

Gray encoding. Our second approach is a variant of binary encoding, namely, Gray encoding [Gray, 1953]. It is a binary encoding system where two successive values differ in only one bit. See examples in Figures 3.2 and 3.3: the same set of binary identifiers are generated, but in a different order compared to the sequence of the attribute values. When combined with PrivBayes, it inherits some of the pros and cons of the natural binary encoding, but potentially brings some advantages. As successive values share most bits, Gray encoding can be more robust to noise. Take the value $(30, 40]$ of attribute "age" as an example (Figure 3.2. If the perturbation in PrivBayes happens to change the first or the last bit of its code 010_2, the noisy code (i.e., 110_2 or 011_2) will correspond to a value adjacent to the original one. Thus, the distortion of information is reduced. In contrast, the natural binary encoding does not have this property. Therefore, Gray encoding is a competitive alternative to the natural binary encoding.

Vanilla encoding. To circumvent the redundancy problem of binary encodings, we propose another approach that keeps all attributes intact during the construction of Bayesian networks. So for an attribute which takes on ℓ different values, instead of trying to encode it with $\lceil \log_2 \ell \rceil$ bits, we directly represent it as a discrete variable with ℓ possible values. Figures 3.2 and 3.3 present two examples of vanilla encoding. Under vanilla encoding, the domain of each attribute is considered to be indivisible. Intuitively, this preserves the semantics of non-binary attributes and avoids generating extra attributes with artificial meaning, which help reduce the uncertainty in learning Bayesian networks.

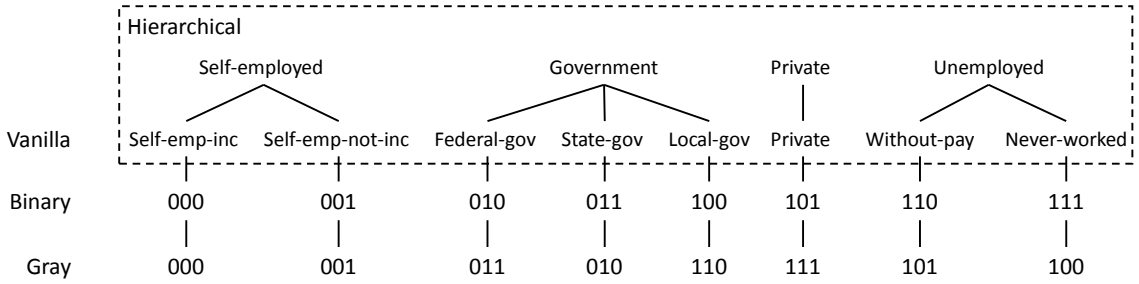


Figure 3.3: Four encodings on a categorical attribute "workclass"

However, applying vanilla encoding on PrivBayes turns out to be challenging. First, we need to revise our notion of θ-usefulness principle and modify the definition of score function F to attributes with general domains. In the following Sections 3.4.2 and 3.4.3, we will clarify how each component of PrivBayes can be updated or redesigned to fit in with non-binary attributes. These effects make the use of non-binary encoding possible within the framework of PrivBayes. Another issue of vanilla encoding is lack of flexibility: attribute inclusion is an "all or nothing" decision. As the domain of each attribute can not be partially encoded, preserving a correlation with a high cardinality can be costly. Recall the "is retired" example in binary encoding. PrivBayes with vanilla encoding has to build a joint distribution of full size 16, because all values of "age" must be present. Due to the θ-usefulness criterion, PrivBayes may choose to discard this distribution of large size when the privacy budget ε is small.

Hierarchical encoding. Last, we present hierarchical encoding, a flexible encoding scheme aware of the semantics of attributes. The main idea is to generalize the domain of each attribute using a taxonomy tree. In Figures 3.2 and 3.3, we illustrate taxonomy trees (roots are omitted) for continuous and categorical attributes, respectively. To each continuous attribute whose domain is partitioned into b bins, we build a binary tree of height $\lceil \log b \rceil$ (excluding the root), where each intermediate node represents the concatenation of bins in its leaves. As for each categorical attribute, we provide a specific taxonomy tree, following the practice in previous work [Iyengar, 2002; Bayardo and Agrawal, 2005]. The leaf nodes depict all the possible values in the domain, then are recursively generalized at the next level of the tree. Take attribute "workclass" in Figure 3.3 as an example. Working for federal, state or local government are three values in the original domain, which can be generalized to working for government at the upper level of the taxonomy tree. It is also worth mentioning that vanilla encoding can be seen as a special case of hierarchical encoding, where each taxonomy tree consists of leaf nodes only.

To apply hierarchical encoding on PrivBayes, we first formalize the concept of generalized attribute. Let height(X) denote the height of attribute X's taxonomy tree.

For each integer $i \in [0, \text{height}(X))$, the nodes at level i (leaves at level 0) of the taxonomy tree define the domain of a generalized version of X, denoted as $X^{(i)}$. The larger i is, the more generalized the attribute will be. Note that $X = X^{(0)}$. With hierarchical encoding, each attribute can provide multiple choices in the construction of Bayesian networks. The tradeoff of attribute generalization is that a less generalized attribute is more informative, yet the more generalized one is more flexible due to its domain of small size. To balance this tradeoff, we resort to the θ-usefulness principle. Following the intuitions in Section 3.3.5, we always prefer the less generalized attributes under the condition that θ-usefulness is guaranteed. In the next section, we will specify how to integrate θ-usefulness with the hierarchical encoding. To avoid a combinatorial explosion of possibilities, we restrict to considering generalizations where we pick a single level i to apply across one attribute, i.e. we do not allow generalizations where different branches of the generalization tree are represented at different levels. Removing this assumption is feasible within our framework, but entails a much larger set of possibilities to search, with a comensurate drain on the privacy budget. Consequently, we choose to make this restriction.

3.4.2 Extension of θ-usefulness

In this subsection, we extend the notion of θ-usefulness to the non-binary encodings, i.e., vanilla and hierarchical encodings. This requires us to handle non-binary attributes (both encodings) and generalized attributes (hierarchical encoding only). For easy understanding, we will first present how each component of PrivBayes can be updated to fit in with non-binary attributes, under the criterion of θ-usefulness; the algorithms for generalized attributes will be elaborated at the end of this subsection.

Non-binary attributes. Recall that, in the proof of Lemma 3.3, all $(k+1)$-dimensional marginals have the same domain size 2^{k+1}, such that we can bound the average probability in each cell by simply limiting k. However, that is no longer the case when the domain sizes of attributes vary. In particular, marginal distributions of the same dimensionality may have very different sizes; for example, the joint distribution of two binary attributes has 4 cells, while that of two attributes with cardinality four each has 16. Therefore, a single k is not sufficient to guarantee θ-usefulness anymore: we should also take the domain size of each attribute into account. Towards this end, we revise Algorithms 3.1 and 3.2 as follows.

First, in Algorithm 3.1, we set $k = 0$, i.e., we materialize all d marginal distributions associated with the Bayesian network $\mathcal{N}$. The privacy budget of this process (i.e., $\varepsilon/2$) is evenly divided into d portions, each of which is spent on a marginal. In other words, the Laplace noise added to each cell of each marginal is $\text{Lap}(\frac{4d}{n\varepsilon})$. Algorithm 3.3 presents the

Algorithm 3.3: NoisyConditionals $(D, \mathcal{N})$: returns $\mathcal{P}^*$

1 initialize $\mathcal{P}^* = \emptyset$;
2 **for** $\underline{i = 1 \textbf{ to } d}$ **do**
3 materialize the joint distribution $\Pr[X_i, \Pi_i]$;
4 generate differentially private $\Pr^*[X_i, \Pi_i]$ by adding noise $\underline{\text{Lap}\left(\frac{4 \cdot d}{n \cdot \varepsilon}\right)}$;
5 set negative values in $\Pr^*[X_i, \Pi_i]$ to 0 and normalize;
6 $\underline{\text{derive } \Pr^*[X_i \mid \Pi_i] \text{ from } \Pr^*[X_i, \Pi_i]; \text{ add it to } \mathcal{P}^*}$;
7 **return** $\mathcal{P}^*$;

Algorithm 3.4: GreedyBayes (D, θ): returns $\mathcal{N}$

1 initialize $\mathcal{N} = \emptyset$ and $V = \emptyset$;
2 randomly select an attribute X_1 from $\mathcal{A}$; add $(X_1, \emptyset)$ to $\mathcal{N}$; add X_1 to V;
3 **for** $i = 2$ **to** d **do**
4 initialize $\Omega = \emptyset$;
5 **foreach** $X \in \mathcal{A} \backslash V$ **do**
6 find all maximal parent sets of X, i.e., $\top(X) = \texttt{MaximalParentSets}\left(V, \frac{n\varepsilon}{4d\theta|\text{dom}(X)|}\right)$;
7 **if** $\top(X) = \emptyset$ **then**
8 add $(X, \emptyset)$ to Ω;
9 **else**
10 **foreach** $\Pi \in \top(X)$ **do** add (X, Π) to Ω;
11 select (X_i, Π_i) from Ω, using the exponential mechanism with privacy budget $\varepsilon/2(d-1)$;
12 add (X_i, Π_i) to $\mathcal{N}$; add X_i to V;
13 **return** $\mathcal{N}$;

revised version of Algorithm 3.1, with the places that have changed underlined to show the difference.

Now consider Algorithm 3.2. Observe that, in Algorithm 3.2, Line 5 is the only place where we interact directly with k, and it generates a candidate set Ω, which contains all eligible attribute-parent pairs for the next round of selection. Specifically, for each attribute $X \in \mathcal{A} \backslash V$, the intention of Line 5 is to help identify every subset Π of V that satisfies the following two requirements:

(i) $\Pr[X, \Pi]$ is θ-useful. The motivation of this requirement is to ensure that the information in $\Pr[X, \Pi]$ will not be overwhelmed by the noise introduced in the distribution learning phase, as we explain in Section 3.3.5;

Algorithm 3.5: MaximalParentSets (V, τ): returns $\mathcal{S}$

```
1 if τ < 1 then return ∅;
2 if V = ∅ then return {∅};
3 pick an arbitrary attribute X from V;
4 initialize S = MaximalParentSets(V \ {X}, τ);
5 foreach Z ∈ MaximalParentSets(V \ {X}, τ / |dom(X)|) do
6     if Z ∈ S then remove Z from S;
7     add Z ∪ {X} to S;
8 return S;
```

(ii) Π is maximal, i.e., there is no subset Π' of V such that $\Pr[X, \Pi']$ is θ-useful and $\Pi \subset \Pi'$. The requirement of maximality of Π relies on the monotonicity of the mutual information I. Given two sets Π and Π' such that $\Pi \subseteq \Pi'$, we always have $I(X, \Pi) \leq I(X, \Pi')$ for any attribute X. Therefore, with respect to the informativeness of the Bayesian network, a larger Π is always preferred.

We refer to a subset Π satisfying the above two conditions as a *maximal parent set* of X. Finding all maximal parent sets of a particular attribute is simple when all attributes are binary. In particular, with Lemma 3.3, one can easily find the appropriate value of k such that all subsets of V with size $\min(k, |V|)$ meet the requirements[4].

However, when some attributes are non-binary, a different calculus applies. Given an attribute-parent pair (X, Π) with m cells in $\Pr[X, \Pi]$, the average probability in each cell is m^{-1}. On the other hand, the average scale of noise is $4d/n\varepsilon$ with our updates in Algorithm 3.3. Therefore, $\Pr[X, \Pi]$ is θ-useful only if $m \leq n\varepsilon/4d\theta$. In other words, given an attribute X, we are only interested in those maximal subsets of V whose domain sizes are no greater than $\frac{n\varepsilon}{4d\theta|\mathrm{dom}(X)|}$.

Based on the above discussion, we present Algorithm 3.4, which is a revised version of Algorithm 3.2. The initialization and outer loop stay the same, but compared to Algorithm 3.2, it adopts a new procedure to generate the candidate set Ω (Lines 5-10). Specifically, for each attribute $X \in \mathcal{A} \backslash V$, we first invoke the MaximalParentSets algorithm (Line 6), which identifies a set $\top(X)$ that contains all maximal parent sets of X. (We will elaborate MaximalParentSets shortly.) After constructing $\top(X)$, we add attribute-parent pair (X, Π) to Ω for each $\Pi \in \top(X)$ (Line 10). In Lines 7-8, we also consider an extreme case that $\top(X)$ is empty, which implies that even $\Pr[X]$ violates θ-usefulness. Following the common practice discussed in Section 3.3.5, we add $(X, \emptyset)$ to Ω in order to ensure that every attribute is modeled (at least as an independent attribute) in the Bayesian network. Once the candidate set Ω is ready to be selected from, we

[4]When $k > |V|$, the only maximal parent set is V itself.

Algorithm 3.6: MaximalParentSets* (V, τ): returns $\mathcal{S}$

```
1  if τ < 1 then return ∅;
2  if V = ∅ then return {∅};
3  pick an arbitrary attribute X from V;
4  initialize S = ∅, U = ∅;
5  for i = 0 to height(X) − 1 do
6      foreach Z ∈ MaximalParentSets*(V \ {X}, τ / |dom(X^(i))|) do
7          if Z ∈ U then continue;
8          add Z to U; add Z ∪ {X^(i)} to S;
9  foreach Z ∈ MaximalParentSets*(V \ {X}, τ) do
10     if Z ∈ U then continue;
11     add Z to S;
12 return S;
```

invoke the exponential mechanism to privately choose an attribute-parent pair, then add it to $\mathcal{N}$ (Lines 11-12).

We now clarify the details of the MaximalParentSets method. Algorithm 3.5 shows the pseudo-code of MaximalParentSets. It takes as input a set of attributes V and an upper bound of domain size τ, and outputs a set $\mathcal{S}$ that contains all maximal subsets of V with domain sizes no larger than τ. The main idea of the algorithm is to recursively build two sets of maximal subsets, with and without a particular attribute $X \in V$, respectively, and then merge them to get the final result. More specifically, we first initialize $\mathcal{S}$ as the set of maximal subsets without attribute X (Line 4). To construct the ones with X, the algorithm utilizes the attribute set $V \backslash X$ and the upper bound $\tau / |\operatorname{dom}(X)|$, to ensure that adding X to each returned subset still guarantees the maximality without violating the restriction on domain sizes. In Lines 6-7, the algorithm updates $\mathcal{S}$ by removing the non-maximal duplications (as Z is a subset of $Z \cup \{X\}$), and then adding the maximal subsets that include the attribute X. Finally, the algorithm has two terminating conditions:

(i) $\tau < 1$. Given that the minimum size of a domain is $\operatorname{dom}(\emptyset) = 1$, this condition indicates that no subset of V meets the restriction on domain size. In that case, the algorithm simply returns the empty set;

(ii) $V = \emptyset$. Given that $\tau \geq 1$ and V is empty, $\emptyset$ is the only subset of V that satisfies the requirement on the domain size. Therefore, the algorithm returns a singleton set $\{\emptyset\}$.

Generalized attributes. Last, we show how to utilize generalized attributes in the framework of PrivBayes. Recall that the intuition of attribute generalization is to

provide high flexibility in the construction of Bayesian networks. In particular, given an attribute-parent pair (X, Π) whose joint distribution $\Pr[X, \Pi]$ violates θ-usefulness, we aim to generalize some attributes in Π to satisfy the restriction on domain size, while preserving as much information between X and Π as possible. Towards this end, we extend the definition of maximal parent sets to allow generalized attributes.

First, we define a *generalized subset* as a subset in which attributes are generalized. Note that the original attribute can be seen as a generalized attribute of itself. Then, for each attribute $X \in \mathcal{A} \backslash V$, the maximal parent set Π is a generalized subset of V that satisfies: (i) $\Pr[X, \Pi]$ is θ-useful; (ii) Π is maximal, i.e., there is no generalized subset Π' of V such that $\Pr[X, \Pi']$ is θ-useful and Π' contains any extra attribute not in Π, or any shared attribute but with a lower generalization level.

By this definition, we update the MaximalParentSets method as in Algorithm 3.6. In Lines 6-8, the algorithm generates maximal parent sets that contain attribute $X^{(i)}$. By utilizing the attribute set $V \backslash X$ and the upper bound $\tau / |\operatorname{dom}(X^{(i)})|$, we ensure that the union of $X^{(i)}$ and each returned subset Z satisfies the requirement on domain size. Before adding $Z \cup \{X^{(i)}\}$ to $\mathcal{S}$, we check its maximality with set $\mathcal{U}$, as $Z \in \mathcal{U}$ implies that the union of Z and a less generalized X is already included in $\mathcal{S}$; therefore, we have to discard the non-maximal $Z \cup \{X^{(i)}\}$. By enumerating generalization levels of X (Line 5), we construct all maximal parent sets with the particular attribute X. In the end, we process the remaining cases in which X is excluded (Lines 9-11).

Although the construction of maximal parent sets is complicated, applying generalized attributes to other parts of PrivBayes is surprisingly simple. No additional change is required in Algorithms 3.3 and 3.4. In both algorithms, the only place where we interact with generalized attributes is to materialize the non-private joint distribution (in order to inject noise and compute score function, respectively). This can be done by generalizing tuples in the input data accordingly. As for data synthesis, the properties of Bayesian networks ensure that each attribute is sampled before appearing in any parent set. Therefore, to sample X from the conditional distribution $\Pr^*[X \mid \Pi]$, all we need is to make sure that the previously sampled attributes in Π are generalized properly.

3.4.3 Alternative Score Function

To facilitate the above extension of θ-usefulness to non-binary encodings, there is one missing piece of the puzzle: we need to identify a score function to be used within the exponential mechanism to select attribute-parent pairs. One natural choice, as in the case of binary domains (see Section 3.3.2), is the mutual information function I in Equation (3.2). Observe that I does not have any restriction on the domain sizes of attributes; therefore, it can be directly applied on general domains. As for the more advanced score function F defined in Section 3.3.3, however, it is not immediately clear

Table 3.4: Properties of score functions

Function	Range	Sensitivity	Time complexity
$I(X, \Pi)$	$O(1)$	$O(\log n/n)$	$O(\lvert \mathrm{dom}(X)\rvert \cdot \lvert \mathrm{dom}(\Pi)\rvert)$
$F(X, \Pi)$	$O(1)$	$O(1/n)$	$O(\lvert \mathrm{dom}(\Pi)\rvert \cdot n^{\lvert \mathrm{dom}(x)\rvert - 1})$
$R(X, \Pi)$	$O(1)$	$O(1/n)$	$O(\lvert \mathrm{dom}(X)\rvert \cdot \lvert \mathrm{dom}(\Pi)\rvert)$

how we may extend F to general domains, since the dynamic programming algorithm for computing F in Section 3.3.4 requires that all input domains are binary. In what follows, we will prove that the extension of F is infeasible, and an alternative score function R will be proposed to work on general domains instead.

First, we observe that computing F is NP-hard in general (proof provided in the Appendix).

Theorem 3.3 *Computing the score function F is NP-hard.*

Given the above NP-hardness result, we know that there is no efficient way to calculate F given an arbitrary joint distribution. Fortunately, we have shown how a pseudo-polynomial dynamic programming solution can be designed to compute F (in Section 3.3.4), by utilizing a special property of PrivBayes. That is, all numbers in the joint distribution are multiples of $1/n$, which provides a possibility to represent the states of dynamic programming at a cost polynomial in n. However, the extension of this solution to general domains has time complexity $O(\lvert \mathrm{dom}(\Pi)\rvert \cdot n^{\lvert \mathrm{dom}(x)\rvert - 1})$; therefore, it is not efficient for any non-binary X.

Recall that F measures the L_1 distance from the input $\Pr[X, \Pi]$ to a joint distribution that *maximizes* $I(X, \Pi)$. The rationale behind this design is two-fold: (i) the L_1 distance measurement guarantees a relatively small sensitivity, i.e., $S(F) = O(1/n)$, and (ii) if $\Pr[X, \Pi]$ is close to a joint distribution of maximum $I(X, \Pi)$, then intuitively, $\Pr[X, \Pi]$ is likely to give a large mutual information between X and Π. Now we borrow the ideas from F, and propose an alternative score function R which can be computed efficiently on non-binary domains. The main difference of R is that it relies on the L_1 distance from $\Pr[X, \Pi]$ to a joint distribution that *minimizes* $I(X, \Pi)$. In the following, we specify the construction of R.

Lemma 3.4 *If $I(X, \Pi) = 0$, then $\Pr[X = x, \Pi = \pi] = \Pr[X = x] \cdot \Pr[\Pi = \pi]$ for any pair of $x \in \mathrm{dom}(X)$ and $\pi \in \mathrm{dom}(\Pi)$.*

The proof is rather straightforward given that $I(X, \Pi) = 0$ implies mutual independence between X and Π, so is omitted here. With Lemma 3.4, one can easily generate a

joint distribution, denoted as $\overline{\Pr}[X, \Pi]$, that satisfies $I(X, \Pi) = 0$. In particular, for any $x \in \mathrm{dom}(X), \pi \in \mathrm{dom}(\Pi)$, we have

$$\overline{\Pr}[X = x, \Pi = \pi] = \Pr[X = x] \cdot \Pr[\Pi = \pi],$$

where $\Pr[X]$ and $\Pr[\Pi]$ are marginal distributions derived from $\Pr[X, \Pi]$. Then, our new score function to evaluating attribute-parent pair (X, Π) is defined as:

$$R(X, \Pi) = \frac{1}{2} \left\| \Pr[X, \Pi] - \overline{\Pr}[X, \Pi] \right\|_1. \tag{3.7}$$

Intuitively, if $R(X, \Pi)$ is small, then $\Pr[X, \Pi]$ must be close to $\overline{\Pr}[X, \Pi]$; therefore, the mutual information between X and Π is likely to be small. On the other hand, a large $R(X, \Pi)$ indicates a large scale of distortion between $\Pr[X, \Pi]$ and $\overline{\Pr}[X, \Pi]$, which implies a strong correlation (and a large I) between X and Π. In short, the value of $R(X, \Pi)$ tends to be positively correlated with $I(X, \Pi)$. In addition, R has a small sensitivity.

Theorem 3.4 $S(R) \leq 3/n + 2/n^2$.

Therefore, we conclude that R is a feasible score function for PrivBayes.

Last, we compare R with its predecessors I and F. Table 3.4 summarizes the properties of the three score functions. Among all choices, F is the most competitive one in term of sensitivity, i.e., $S(F)$ is less than 1/3 of $S(R)$ (comparing the constants in Theorems 3.2 and 3.4) and both of them are much smaller than $S(I)$. This makes F preferable to the other two functions, but only in the case of binary domains. As for non-binary attributes, R is the score function of first choice. In the experimental section, we will empirically evaluate the performance of different score functions in constructing Bayesian networks.

3.5 Experiments

3.5.1 Experimental Settings

Datasets. We make use of four real datasets in our experiments: (i) NLTCS [Manton, 2010], which contains records of 21,574 individuals participated in the National Long Term Care Survey, (ii) ACS [Ruggles et al., 2015], 47,461 rows of personal information obtained from 2013 and 2014 ACS sample sets in IPUMS-USA, (iii) Adult [Bache and Lichman, 2013], which includes the information of 45,222 individuals extracted from the 1994 US Census, and (iv) BR2000 [Ruggles et al., 2015], which consists of 38,000 census records collected from Brazil in year 2000. The first two datasets contain only binary attributes, while the last two have both continuous and categorical attributes. Note that

Table 3.5: Dataset characteristics

Dataset	Cardinality	Dimensionality	Domain size
NLTCS	$21,574$	16	$\approx 2^{16}$
ACS	$47,461$	23	$\approx 2^{23}$
Adult	$45,222$	15	$\approx 2^{52}$
BR2000	$38,000$	14	$\approx 2^{32}$

the domain of each continuous attribute is discretized into 16 equi-width bins. Table 3.5 illustrates the properties of the datasets.

Tasks. We evaluate the performance of PrivBayes on two different tasks. The first task is to build all α-way marginals of a dataset [Barak et al., 2007]. For convenience, we use Q_α to denote the set of all α-way marginals. We measure the accuracy of each noisy marginal by the *total variation distance* [Cybakov, 2009b] between itself and the noise-free marginal (i.e., half of the L_1 distance between the two marginals, when both of them are treated as probability distributions). We use the average accuracy over all marginals as the final error metric for Q_α.

The second task that we consider is to *simultaneously* train multiple SVM classifiers on a dataset, where each classifier predicts one attribute in the data based on all other attributes. Specifically, on NLTCS, we construct four classifiers to predict whether a person (i) is unable to get outside, (ii) is unable to manage money, (iii) is unable to bathe, and (iv) is unable to travel, respectively. Meanwhile, on ACS, we train four classifiers to predict whether an individual (i) owns a private dwelling, (ii) has a mortgage loan, (iii) lives in a multi-generation family, and (iv) attends school, respectively. On Adult, four classifiers are trained to predict whether an individual (i) is a female, (ii) makes over $50K$ a year, (iii) holds a post-secondary degree, and (iv) has never married, respectively. Last, on BR2000, we train four classifiers to predict whether an individual (i) is a Catholic, (ii) owns at least one car, (iii) has at least one child, and (iv) is older than 20, respectively. For each classification task, we use 80% of the tuples in the data as the training set, and the other 20% as the testing set. We apply PrivBayes on the training data to generate a synthetic dataset, and then use the synthetic data to construct SVM classifiers. The quality of each classifier is measured by its *misclassification rate* on the testing set, i.e., the fraction of tuples in the testing data that are incorrectly classified.

For each of the aforementioned tasks, we repeat each experiment on each method 100 times, and we report the average measurements in our experimental results.

Baselines. For the task of answering count queries in Q_α, we compare PrivBayes with three approaches: (i) *Laplace* [Dwork et al., 2006b], which generates all α-way marginals of a dataset and then injects Laplace noise directly into each cell of the marginals, (ii) *Fourier* [Barak et al., 2007], which transforms the input data D into the Fourier domain,

adds Laplace noise to a subset of Fourier coefficients and uses the noisy coefficients to construct α-way marginals, and (iii) *Contingency*, which first builds the noisy contingency table, and then projects it onto attribute subsets to compute marginals. However, *Contingency* is only applicable to NLTCS and ACS since its computational cost is proportional to the domain size of input data. We also considered several other existing approaches [Hardt et al., 2012; Li et al., 2010; Li and Miklau, 2012; Yuan et al., 2012; Ding et al., 2011] for answering count queries under differential privacy, but find them inapplicable due to our datasets' large domain size. For fair comparison, we adopt two consistency techniques to boost the accuracies of baselines: (i) non-negativity, which rounds all negative counts in a noisy marginal to 0, and (ii) normalization, which linearly rescales the counts in a noisy marginal to make them sum to n.

For the task of training multiple SVM classifiers, we compare PrivBayes against four methods: *PrivateERM* [Chaudhuri et al., 2011], *PrivGene* [Zhang et al., 2013], *NoPrivacy*, and *Majority*. In particular, *PrivateERM* and *PrivGene* are two state-of-the-art methods for SVM classification under differential privacy. *NoPrivacy* constructs classifiers directly on the input data without any privacy protection. *Majority* is a naïve classification method under differential privacy that works as follows. Let $Y = \{0, 1\}$ be the attribute to be classified, and n be the number of tuples in the training data. *Majority* first counts the number of tuples in the training data with $Y = 1$, and then adds Laplace noise (with scale $1/\varepsilon$) into the count to ensure ε-differential privacy. If the noisy count is larger than $n/2$, then *Majority* predicts that all tuples in the testing data should have $Y = 1$; otherwise, *Majority* predicts $Y = 0$ for all tuples. For PrivBayes, *PrivGene*, and *NoPrivacy*, we adopt the standard hinge-loss C-SVM model [Chang and Lin, 2011] with $C = 1$; for *PrivateERM*, we adopt a slightly different SVM model with Huber loss [Chaudhuri et al., 2011], as it does not support the hinge-loss model.

3.5.2 Effect of Score Functions

In the first set of experiments, we evaluate the effectiveness of score functions F (in Section 3.3.3) and R (in Section 3.4.3), against the mutual information function I. Figure 3.4 illustrates the performance of PrivBayes when combined with F, R and I, respectively. The performance of each combination is evaluated by the sum of the mutual information of every attribute-parent pair in the Bayesian network $\mathcal{N}$, i.e., $\sum_{i=1}^{d} I(X_i, \Pi_i)$. The parameter of θ-usefulness is set to 3.

For binary datasets NLTCS and ACS, observe that F and R consistently outperform I in all cases. This is consistent with our analysis in Sections 3.3.3 and 3.4.3 that these advanced score functions help improve the quality of the Bayesian network constructed by PrivBayes. Compare score functions F and R. They achieve almost identical results on large ε (e.g., $\varepsilon \geq 0.4$), while F becomes preferable on small ε. The reason is that

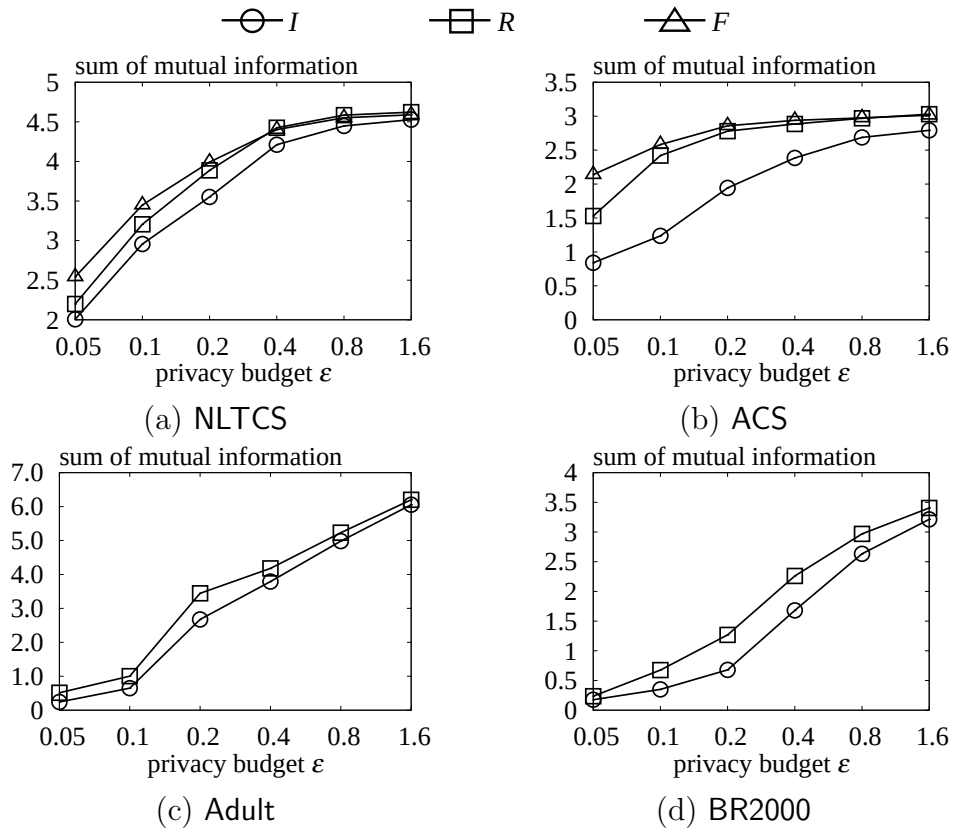
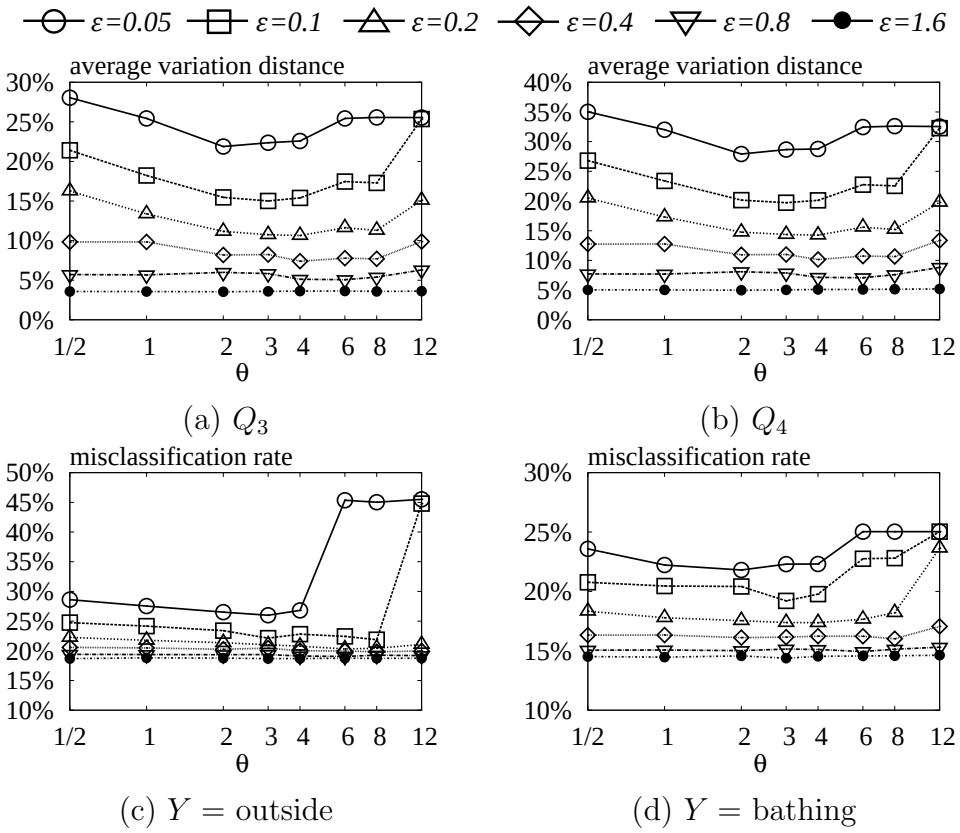


Figure 3.4: Comparison among different score functions

F and R share the same range (e.g., 0.5 for binary attributes), but the sensitivity of F is only 1/3 of that of R. This leads to better utility of F especially when the scale of perturbation in selecting attribute-parent pairs is large. In short, F is the best score function for PrivBayes when all attributes are binary.

On datasets Adult and BR2000, we adopt a different setting in order to test the performance of score functions in non-binary cases. In particular, the vanilla encoding is applied on both continuous and categorical attributes of these datasets. As a consequence, we have to omit F because it is hard to compute in general domains (see proof in Section 3.4.3). The experimental results again support the superiority of R to I. Therefore, we adopt R as the score function of PrivBayes, when non-binary attributes are included.

Regarding the efficiency of computing score functions, F is significantly more time-consuming than the other two, since it takes $O(n2^k)$ for one attribute-parent pair. Functions R and I, in contrast, only take $O(2^k)$. For small k values (i.e., $k = 0, 1, 2$), the time taken to construct a Bayesian network with F is less than 1 minute and is negligible. For larger values of k, the time taken is typically higher, e.g., a few hours in the case of $k = 6$



(a) Q_3 (b) Q_4

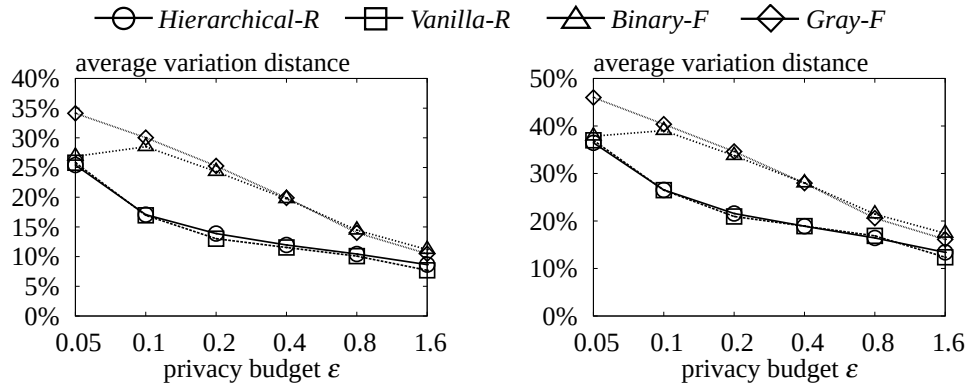
(c) Y = outside (d) Y = bathing

Figure 3.5: Choice of θ on NLTCS

on NLTCS and about 40 minutes for $k = 4$ on ACS. Note that this does not represent a major concern for the applicability of these methods, since we do not consider data release to be a real-time problem. The computation of F for different combinations of attributes can be easily parallelized if higher levels of performance are required.

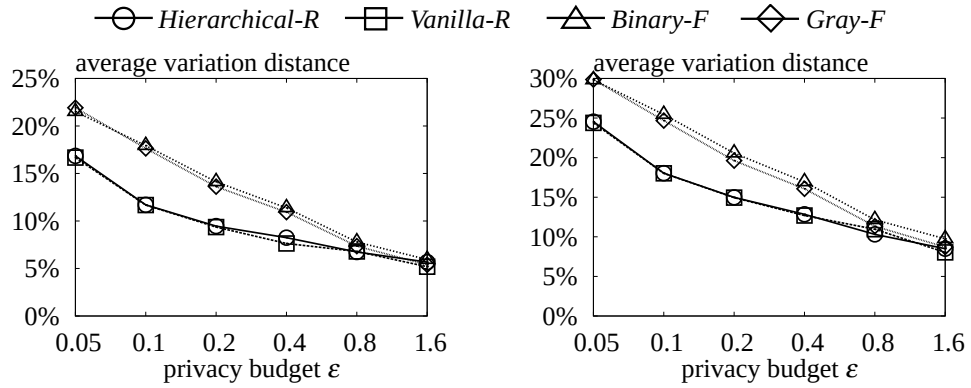
3.5.3 Choice of θ

Recall that PrivBayes has only one internal parameter: the usefulness of noisy marginals θ. As discussed in Section 3.3.5, we adopt the θ-usefulness criterion to automatically select an appropriate value for k, based on the number of tuples in the input data D as well as the domains of the attributes in D. To evaluate the effect of θ on PrivBayes, we adopt four queries on NLTCS, i.e., counting queries Q_3, Q_4 and classification queries to predict disabilities on getting outside and bathing. After that, we examine the performance of PrivBayes in answering these queries, with varying θ and ε. Figure 3.5 illustrates the results. Observe that the error (the average variation distance on counting or the misclassification rate on classification) tends to be higher when θ is very small or very large. This is consistent with our analysis in Section 3.3.5 that



(a) Adult, Q_2 (b) Adult, Q_3

Figure 3.6: Comparison among different encodings (α-way marginals on Adult)

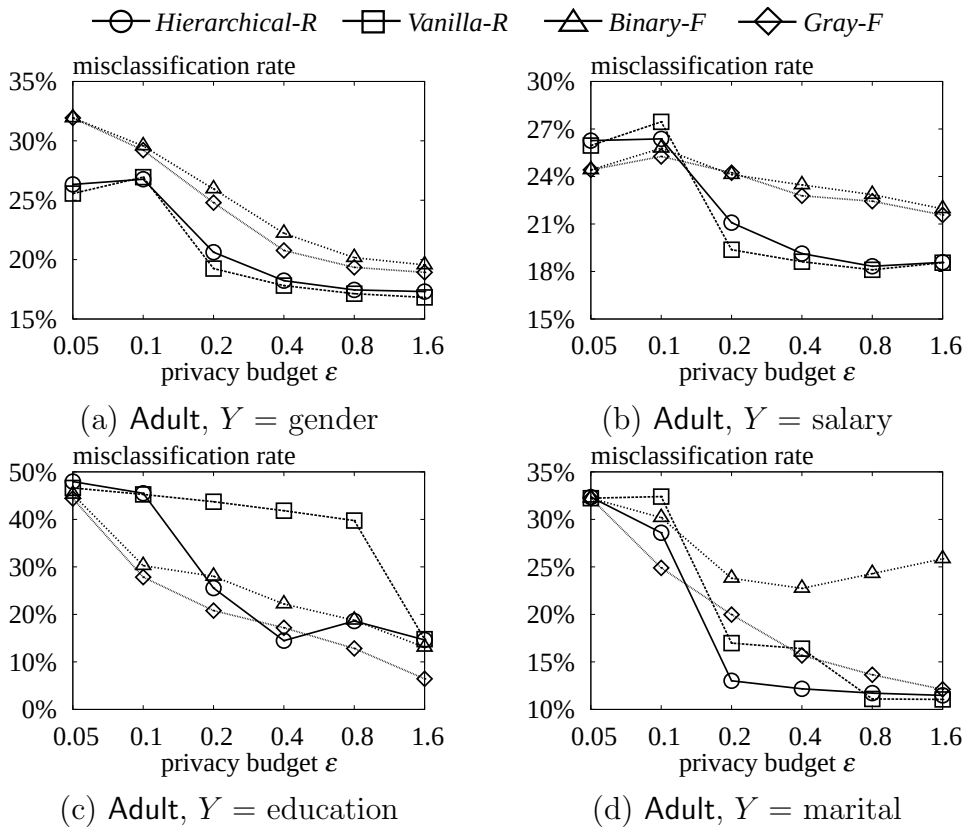


(a) BR2000, Q_2 (b) BR2000, Q_3

Figure 3.7: Comparison among different encodings (α-way marginals on BR2000)

(i) small θ leads to very noisy marginal distributions in the second phase of PrivBayes, which makes the synthetic dataset drastically different from the input data, and (ii) large θ makes it difficult for PrivBayes to construct a high quality Bayesian network in its first phase, which also leads to inferior synthetic data. Based on Figure 3.5, we infer that an appropriate value for θ should be in the range of $[2, 4]$. For all subsequent experiments, we set $\theta = 3$.

Note that our tuning of θ is only based on a portion of results from NLTCS, without inspecting other datasets or other tasks on NLTCS. Therefore, our choice of θ does not reveal any private information on ACS, Adult or BR2000, so it will not unfairly favor PrivBayes on any task on these datasets.



(a) Adult, Y = gender

(b) Adult, Y = salary

(c) Adult, Y = education

(d) Adult, Y = marital

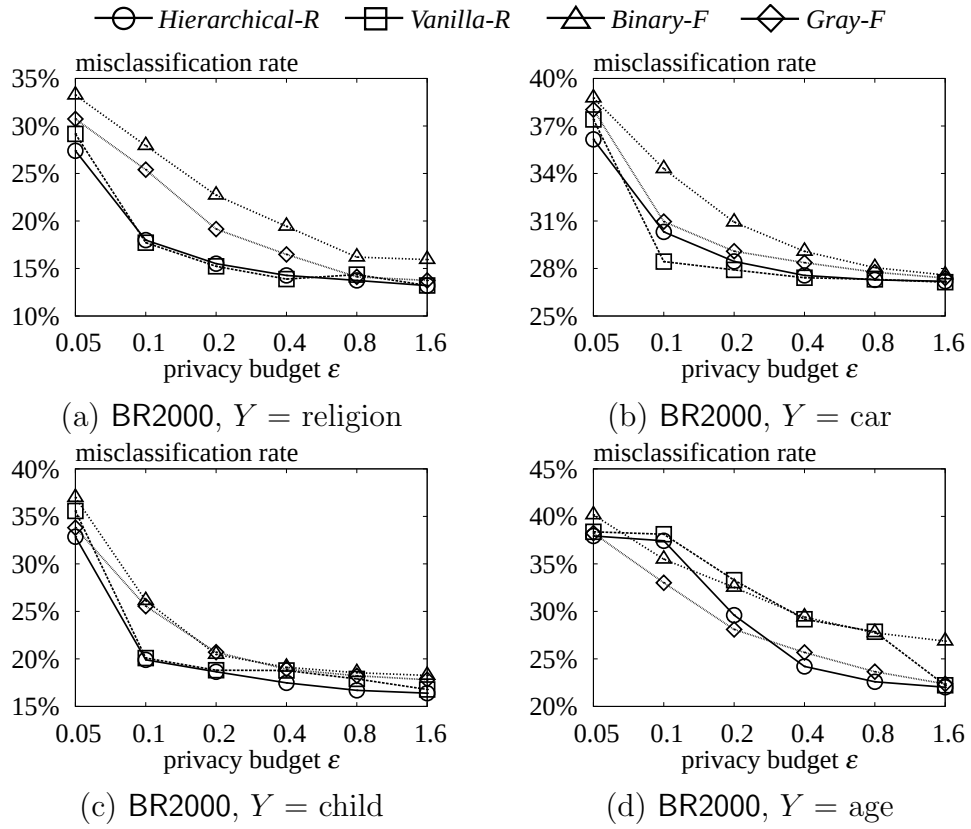
Figure 3.8: Comparison among different encodings (multiple SVM classifiers on Adult)

3.5.4 Encodings on Non-binary Attributes

As discussed in Section 3.4.1, we implement four encodings for datasets with non-binary attributes. Figures 3.6—3.9 show all experimental results over count and classification tasks, on non-binary datasets Adult and BR2000. The name of each reported method indicates the type of encoding and the score function used in the exponential mechanism, e.g., Hierarchical-R stands for the hierarchical encoding and the score function R.

On the results of count queries in Figures 3.6 and 3.7, non-binary encodings are clearly superior to binary ones when ε is small, but the gap shrinks as ε increases. It implies that, in the binary encodings, the information loss caused by the redundant attributes overwhelms the gain of flexible encoding, especially when the perturbation in networking learning is large. This matches our analysis on the drawbacks of binary encoding in Section 3.4.1. Between two non-binary encodings, Hierarchical-R and Vanilla-R achieve almost identical results, which again proves that the count queries in low-dimensional marginals are not sensitive to the flexibility of encoding.

The second set of tasks that we evaluate is training SVM classifiers (see Figures 3.8 and 3.9). In summary, Hierarchical-R achieves the best overall performance among all



(a) BR2000, Y = religion (b) BR2000, Y = car

(c) BR2000, Y = child (d) BR2000, Y = age

Figure 3.9: Comparison among different encodings (multiple SVM classifiers on BR2000)

methods, and its performance is quite stable over different tasks. The reason is that Hierarchical-R is the only method that provides flexible encoding while preserving the semantics of attributes. Take the tasks in Figures 3.8(a) and 3.8(c) as examples. The former task is to predict the gender of an individual. As the predicted attribute is the most flexible in nature (a binary attribute), the advantage of flexible encoding becomes insignificant. As a consequence, the encodings aware of attribute semantics are more preferable. In contrast, the latter task requires to predict the value of a categorical attribute of large domain size, for which Vanilla-R fails to preserve any correlation until $\varepsilon > 0.8$.

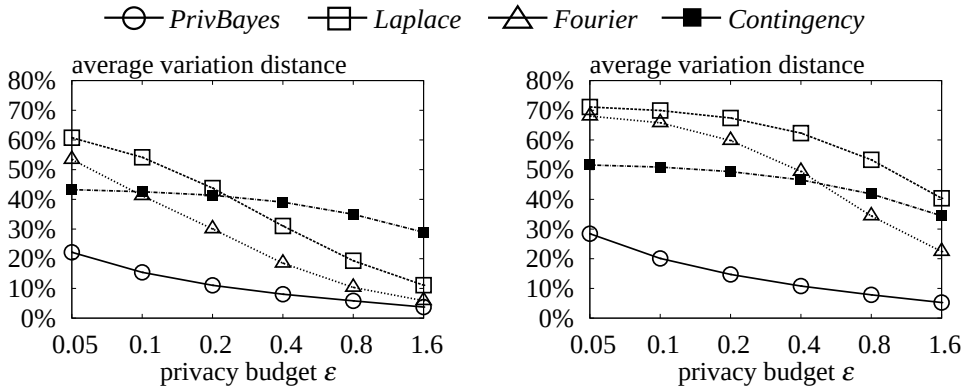
To conclude, we strongly recommend to use the hierarchical encoding on datasets with general domains. In the rest of experiments, we use Hierarchical-R as the routine of PrivBayes on datasets Adult and BR2000.

3.5.5 α-way Marginals

This section compares PrivBayes with the *Laplace*, *Fourier* and *Contingency* approaches on multiple sets of marginals over four datasets.

Table 3.6: Running time in seconds

datasets	queries	PrivBayes	Laplace	Fourier	Contingency
NLTCS	Q_2	8.658	0.990	0.569	2.201
	Q_3	12.17	5.916	2.956	11.92
	Q_4	57.64	26.65	12.92	51.78
ACS	Q_2	317.4	4.931	2.964	46.68
	Q_3	395.7	44.82	24.16	126.3
	Q_4	604.1	300.9	160.0	417.0
Adult	Q_2	6.718	2.155	347.4	n/a
	Q_3	24.58	11.37	12507	n/a
	Q_4	113.6	55.56	>12h	n/a
BR2000	Q_2	11.64	1.297	48.51	n/a
	Q_3	22.05	6.791	1324	n/a
	Q_4	61.35	27.03	20317	n/a

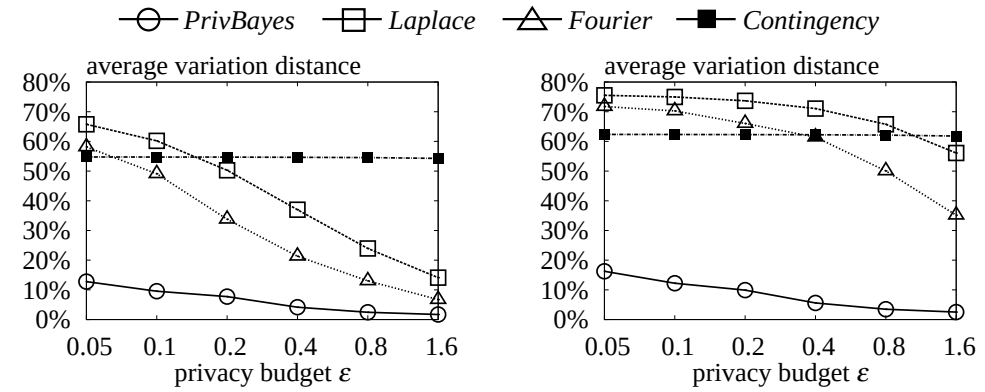


(a) NLTCS, Q_3 (b) NLTCS, Q_4

Figure 3.10: Comparison to baselines (α-way marginals on NLTCS)

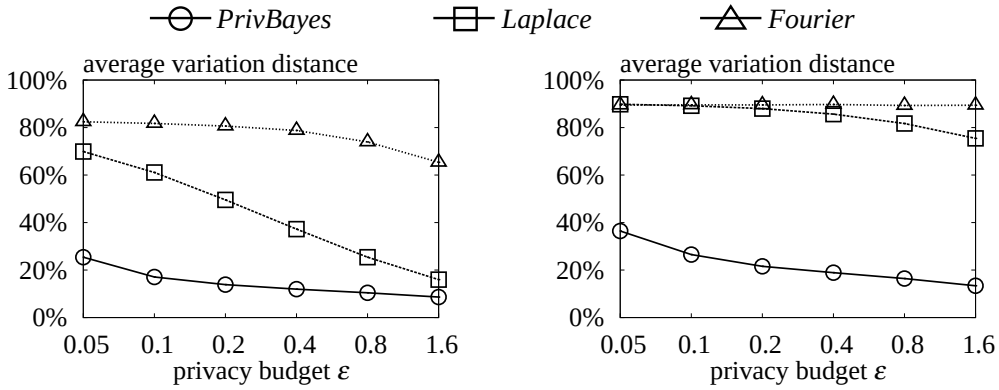
Running Time. Table 3.6 illustrates the average running time of each method at $\varepsilon = 0.1$, varying marginals of datasets. Each number reported consists of the time spent in loading data, building the model and answering the query set. As mentioned in experimental settings, *Contingency* is only applicable to datasets of small domain size; therefore, we only report its running times on NLTCS and ACS.

Observe that the computation costs of all methods increase with the value of α. This is reasonable as a set of higher degree marginals requires more time in query answering and/or model building. *Laplace* is always among the most efficient ones in all cases, as it only employs simple computations: building the marginals, and inserting Laplace noise into each cell of marginals. To PrivBayes and *Contingency*, the model building is independent to query sets, but query answering costs more when α increases. *Fourier* provides some interesting results: it is the most efficient method on binary datasets, while



(a) ACS, Q_3 (b) ACS, Q_4

Figure 3.11: Comparison to baselines (α-way marginals on ACS)



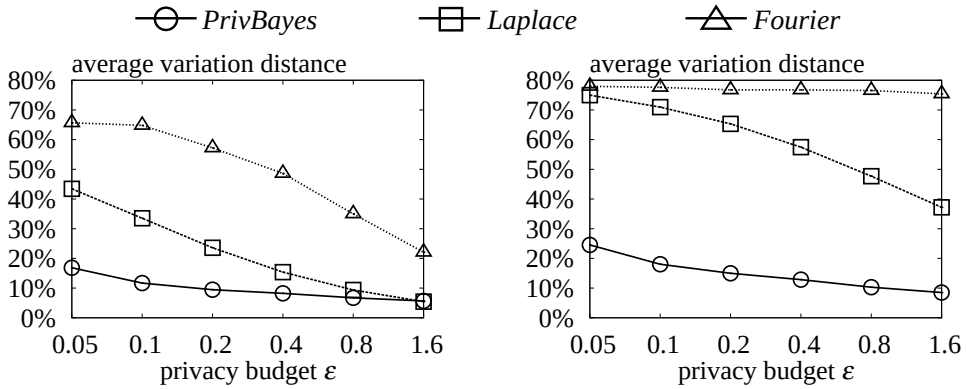
(a) Adult, Q_2 (b) Adult, Q_3

Figure 3.12: Comparison to baselines (α-way marginals on Adult)

performing badly on non-binary ones especially when α is large. To explain, recall that *Fourier* requires all attributes of the data to be binary; therefore, each non-binary attribute has to be converted into a set of binary ones (see binary encoding in Section 3.4.1). This significantly increases the complexity of building marginals of high degree. For example, a 4-way marginal of hexadecimal attributes will be transformed into a 16-way marginal after binarization. Building such a marginal requires to generate 2^{16} Fourier coefficients, which is extremely time-consuming. In Table 3.6, we observe that *Fourier* takes more than 12 hours (resp. 5 hours) to answer query Q_4 of Adult (resp. BR2000), which is inapplicable if the experiment requires hundreds of repeats.

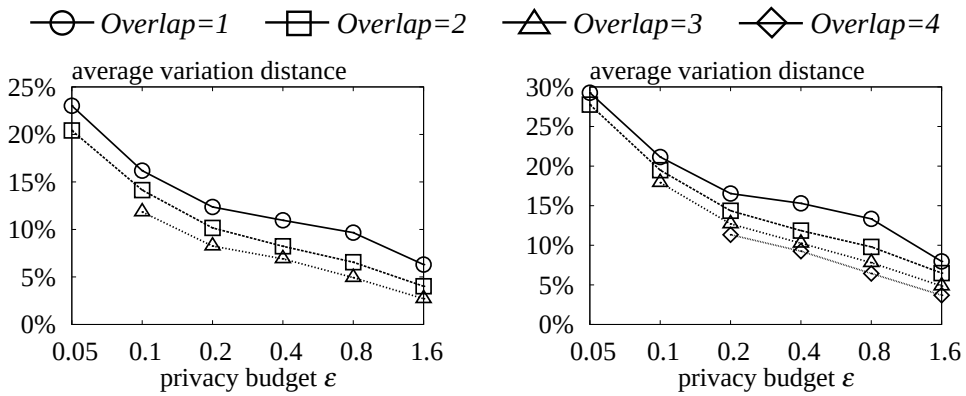
In the following experiments, we evaluate Q_3 and Q_4 on binary datasets NLTCS and ACS, but examine Q_2 and Q_3 instead on the non-binary datasets Adult and BR2000, in order to include *Fourier* in comparison.

Average Error. Figures 3.10—3.13 show the average variation distance of each method



(a) BR2000, Q_2 (b) BR2000, Q_3

Figure 3.13: Comparison to baselines (α-way marginals on BR2000)

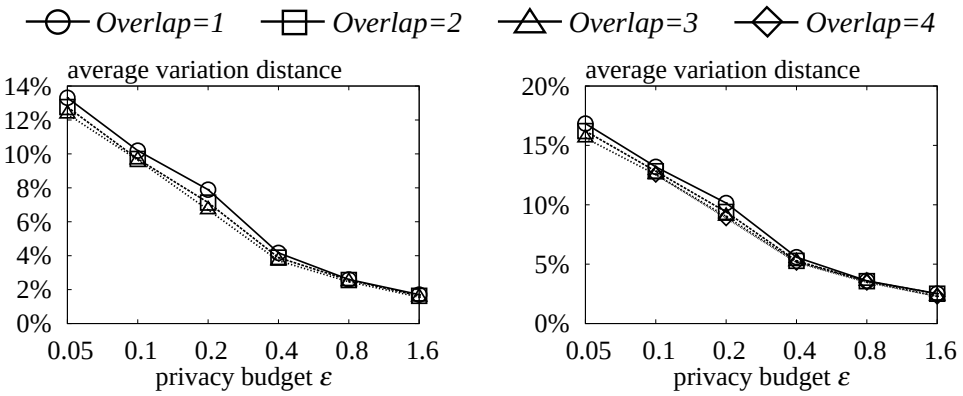


(a) NLTCS, Q_3 (b) NLTCS, Q_4

Figure 3.14: Marginals grouped by overlap (on NLTCS)

for each query set Q_α, varying the privacy budget ε. PrivBayes clearly outperforms the other three baselines in all cases. The relative superiority of PrivBayes is more pronounced when (i) ε decreases or (ii) the value of α increases. To explain, observe that when ε is small, PrivBayes chooses to construct a very low-degree Bayesian network (down to $k = 0$), due to the θ-usefulness criterion. As a consequence, the marginal distributions in the second phase of PrivBayes will be more robust against noise injection, which ensures the quality of the synthetic data will not degrade too significantly. In contrast, the performance of *Laplace* and *Fourier* is highly sensitive to ε, owing to which they incur considerable errors when ε decreases. *Contingency* also generates inaccurate marginals with low privacy budget, with performance improving slightly as ε increases.

Meanwhile, when α increases, the query set Q_α corresponds to a larger set of marginals, in which case the queries Q_α have a higher sensitivity. Therefore, *Laplace* needs to inject a larger amount of noise into Q_α for privacy protection, leading to higher



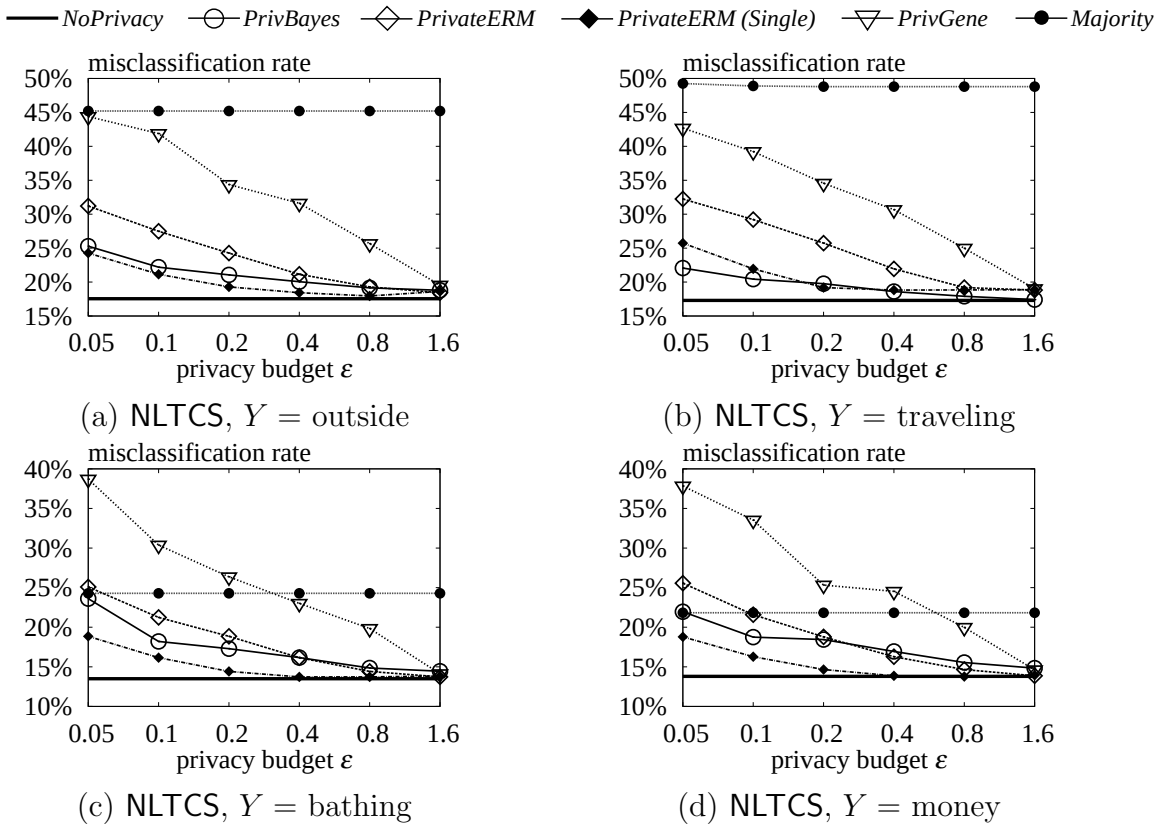
(a) ACS, Q_3 (b) ACS, Q_4

Figure 3.15: Marginals grouped by overlap (on ACS)

query errors. *Fourier* also suffers from a similar issue. On the other hand, the error of PrivBayes is not sensitive to α, as the Bayesian network constructed (once) by PrivBayes enables it to nicely capture the correlations among attributes. Consequently, it can closely approximate the marginals pertinent to Q_α even when α increases.

Additional Experiments. This set of experiments aims to study the impact of two error sources in PrivBayes: the error introduced by the Laplace noise (as in Algorithm 3.1) and the error occurred in Bayesian network modeling. Recall that PrivBayes approximates the original data, via a set of noisy marginals selected by Bayesian networks. Let $\mathcal{M}$ denote this set of marginals. Since marginals in $\mathcal{M}$ are directly produced by Algorithm 3.1, the only source of error is the Laplace noise. In contrast, other marginals of the data are estimated from the Bayesian network and noisy marginals in $\mathcal{M}$, and thus having more loss in terms of accuracy. To illustrate, we group marginals in Q_α by their overlaps with marginals in $\mathcal{M}$. More specifically, we measure the maximum number of attributes that a marginal in Q_α shares with one of the marginals in $\mathcal{M}$. Figures 3.14—3.15 present the average variation distance of marginals grouped by overlaps, on two binary datasets NLTCS and ACS. Note that some points are omitted, as the degree of Bayesian network is too small to support these results. For example, when $\varepsilon = 0.05$, PrivBayes builds 1-degree Bayesian networks over NLTCS dataset; therefore, no overlap of three attributes can be observed.

The results in Figures 3.14—3.15 are consistent with our analysis that the less a marginal overlaps with marginals in $\mathcal{M}$, the more error we shall observe. In all cases, the marginals with the minimum overlap (i.e., $Overlap = 1$) present the largest scale of perturbation, while the ones with maximum overlap (i.e., $Overlap = 3$ for Q_3 and $Overlap = 4$ for Q_4) performing the best. Interestingly, the relative superiority of marginals with large overlaps is more pronounced in NLTCS than ACS. This reflects the



(a) NLTCS, Y = outside

(b) NLTCS, Y = traveling

(c) NLTCS, Y = bathing

(d) NLTCS, Y = money

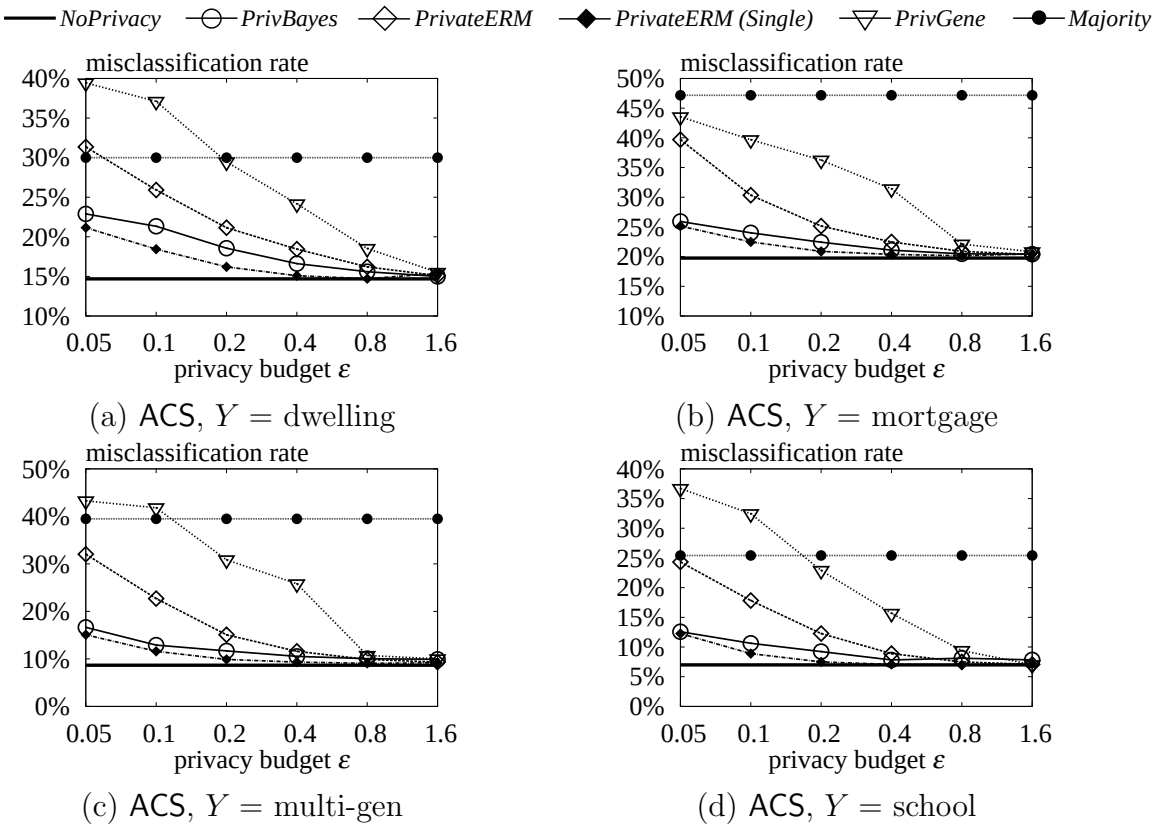
Figure 3.16: Comparison to baselines (multiple SVM classifiers on NLTCS)

fact that Bayesian networks built over NLTCS introduce more errors into those estimated marginals of the data.

3.5.6 Multiple SVM Classifiers

In the last set of experiments, we evaluate different methods for SVM classification. As explained in Section 3.5.1, on each dataset, we train four SVM classifiers simultaneously. For PrivBayes, we apply it to generate only *one* synthetic dataset D^* from each training set, and then use D^* to train all four classifiers required. The other differentially private methods (i.e., *PrivateERM*, *PrivGene*, and *Majority*) can only produce one classifier at a time. Therefore, for each of those method, we evenly divide the privacy budget ε into four parts, and use $\varepsilon/4$ budget to train each classifier. To illustrate the performance of *PrivateERM* when building a single classifier, we include an additional baseline referred to as *"PrivateERM (Single)"*. This baseline is identical to *PrivateERM*, expect that it uses a privacy budget of ε (instead of $\varepsilon/4$) in training each classifier.

Figures 3.16—3.19 show the misclassification rate of each method as a function of the overall ε. The error of *NoPrivacy* remains unchanged for all ε, since it does not enforce



(a) ACS, Y = dwelling

(b) ACS, Y = mortgage

(c) ACS, Y = multi-gen

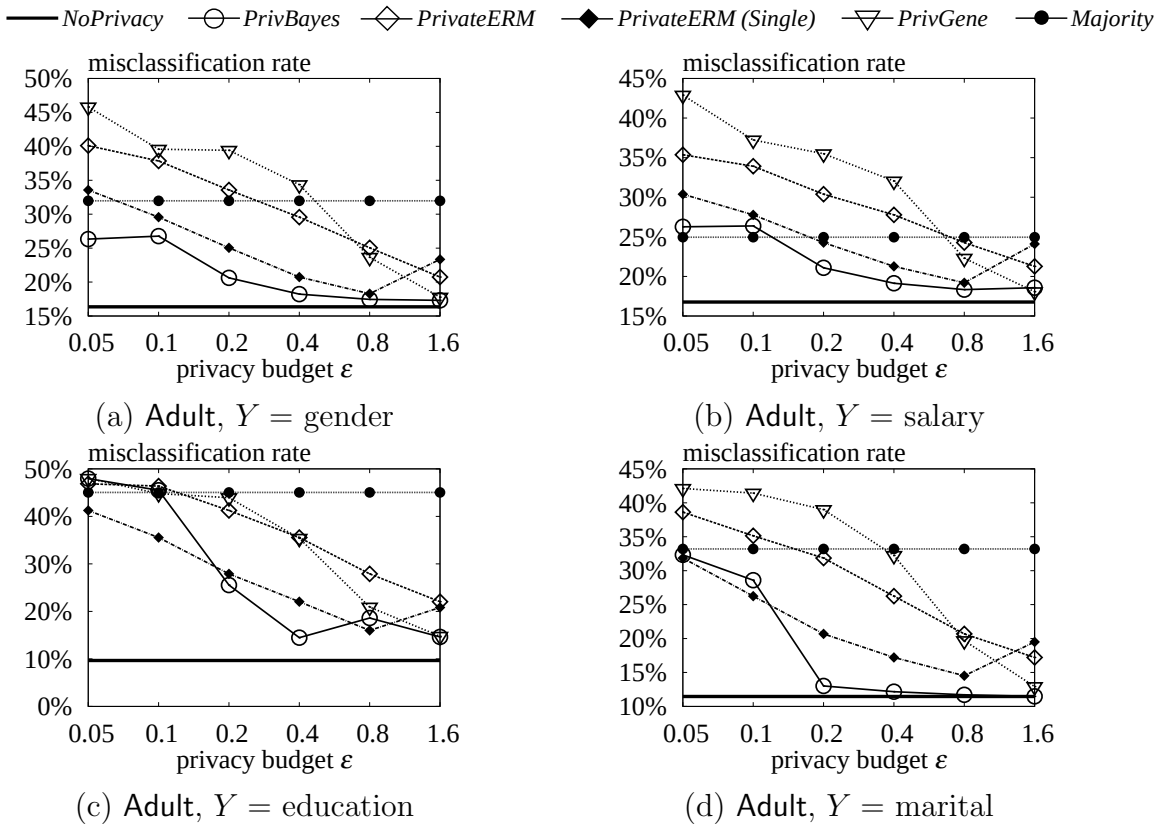
(d) ACS, Y = school

Figure 3.17: Comparison to baselines (multiple SVM classifiers on ACS)

ε-differential privacy—it represents the best case to aim for. The accuracy of *Majority* is insensitive to ε, since (i) it performs classification only by checking whether there exists more than 50% tuples in the training set with a certain label, and (ii) this check is quite robust against noise injection when the number of tuples in the training set is large (as is the case in our experiments). As for the other methods, PrivBayes consistently outperforms *PrivateERM* and *PrivGene* in almost all cases, except for a few settings in Figures 3.16(d) and 3.19(d). Interestingly, in Figure 3.18(c), the misclassification rate of PrivBayes increases when ε changes from 0.4 to 0.8. The reason is that we have tuned the parameter θ for PrivBayes based on queries on NLTCS, and hence, our choice of θ does not always guarantee the best performance for PrivBayes on other datasets. Overall, PrivBayes is superior to both *PrivateERM* and *PrivGene* on the classification task.

On the other hand, PrivBayes is outperformed by *PrivateERM (Single)*[5] in most cases (except on Adult). This is reasonable given that *PrivateERM* is designed solely for

[5]The behavior of *PrivateERM (Single)* on Adult with $\varepsilon = 1.6$ is an artifact of the algorithm itself: it computes an internal parameter ε'_p as a function of ε, which yields a sub-optimal choice when $\varepsilon = 1.6$.



(a) Adult, Y = gender

(b) Adult, Y = salary

(c) Adult, Y = education

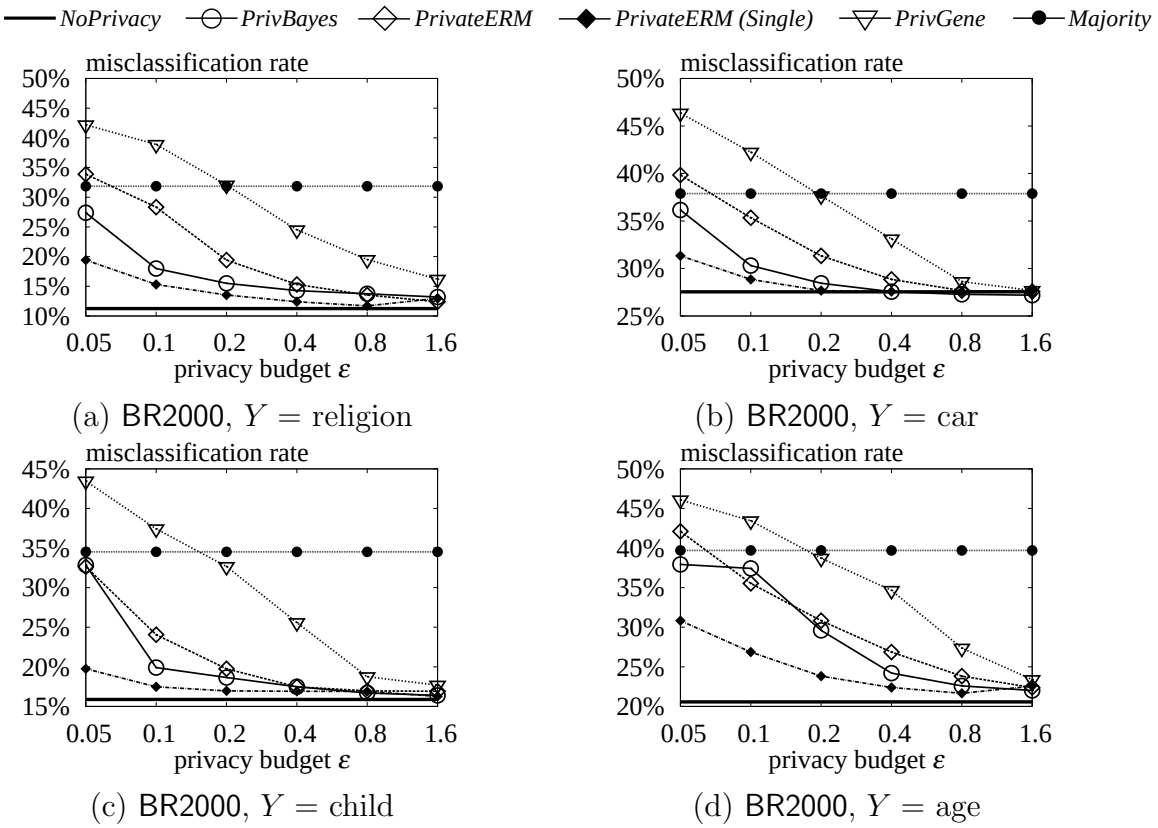
(d) Adult, Y = marital

Figure 3.18: Comparison to baselines (multiple SVM classifiers on Adult)

SVM classification, whereas PrivBayes does not specifically optimize for SVM classification when it generates the synthetic data. In general, the fact that PrivBayes can support multiple analytical tasks (without incurring extra privacy overhead) makes it highly favorable in the common case when the user does not have a specific task in mind and would like to conduct *exploratory* data analysis by experimenting with various tasks.

Parameter Tuning. The privacy-preserving SVM training algorithms presented so far assume that the regularization parameter C of C-SVM is provided as an input. However, C is usually unknown in real applications and should be carefully selected based on the training data. In this set of experiments, we show how PrivBayes and PrivateERM tune the value of C in a differentially private manner, then compare their performance.

As PrivBayes releases a synthetic training dataset D^*, it is straightforward to tune the best value of C by applying the standard 5-fold cross-validation algorithm [Chang and Lin, 2011] on D^*. We denote this approach by PrivBayes (tune). Another differentially private parameter tuning technique in use is a variant of PrivateERM presented in [Chaudhuri et al., 2011]. This approach, dubbed PrivateERM (tune), uses the exponential mechanism [McSherry and Talwar, 2007] by training classifiers with different



(a) BR2000, Y = religion

(b) BR2000, Y = car

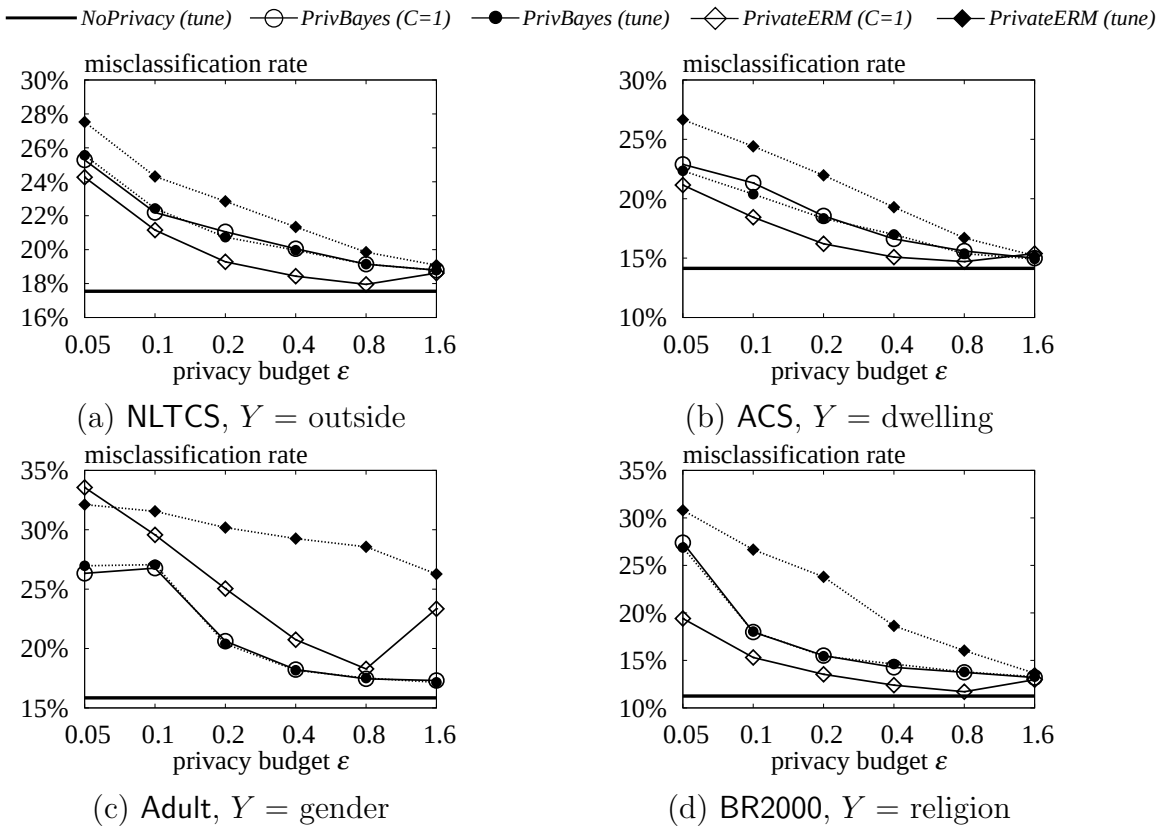
(c) BR2000, Y = child

(d) BR2000, Y = age

Figure 3.19: Comparison to baselines (multiple SVM classifiers on BR2000)

values of C on *disjoint subsets* of the training data, then releasing one model with a randomized selection. We also include two methods in previous experiments for comparison: PrivBayes (C=1) and PrivateERM (C=1), both using a fixed value $C = 1$. In addition, NoPrivacy (tune), a non-private SVM classifier with a tuned C, is reported to represent the best case to aim for. For each method, we provide an exponentially growing sequence of C for selection, from 10^{-3} to 10^3 with common ratio 10.

Figure 3.20 shows the empirical results on four tasks. Note that the performance of PrivBayes (tune) is close to that of PrivBayes (C=1) in most cases, which implies that the default value $C = 1$ is luckily among the best options. Some improvements can be observed in Figure 3.20(b) when $\varepsilon < 0.2$, in which PrivBayes (tune) selects small values of C (mostly 0.01) for better performance. In contrast, the gap between PrivateERM (tune) and PrivateERM (C=1) is significant, and interestingly, the tuning version performs worse. To explain, recall that PrivateERM (tune) trains for different values of C on separate subsets of the training data. As a consequence, each subset contains only a small portion of the data, and becomes less resilient to noise. Although multiple models of C-SVM are built and tested, the quality of each tends to be rather poor; thus, the output is of low utility. Therefore, we conclude that PrivBayes (tune) is



(a) NLTCS, Y = outside

(b) ACS, Y = dwelling

(c) Adult, Y = gender

(d) BR2000, Y = religion

Figure 3.20: Experiments on parameter tuning

a more preferable parameter tuning solution than PrivateERM (tune) in training private SVM classifers.

3.6 Discussions

We have seen that PrivBayes is a highly effective model to release data while respecting privacy. In this section, we discuss its major limitation: PrivBayes requires to discretize the domain of each continuous attribute into a fixed number b of equi-width bins. In our experiments, we adopt an ad hoc value 16. However, the choice of b is highly non-trivial. Intuitively, a small b makes the resulting marginals too coarse-grained but robust against noise. On the other hand, a large b leads to less information loss in discretization, but excessive noise in the private marginals. One potential solution is to enable the tuning of b in the learning process of Bayesian networks. This requires to design a new score function that includes a penalty on large b, such that the tradeoff between bias (caused by small b) and variance (caused by large b) can be balanced.

Chapter 4

PrivTree: Private Release of Spatial and Sequential Data

This chapter presents PrivTree, an algorithm for the decomposition problem that adopts the hierarchical approach but completely eliminates the dependency on the maximum depth of recursion when splitting the domain. In particular, PrivTree requires only *a constant amount of noise* in deciding whether a sub-domain should be split, which enables it to generate fine-grained decompositions without worrying about the recursion depth. Such a surprising improvement is obtained with a novel mechanism for differential privacy that exploits a non-trivial analysis on the Laplace noise [Dwork et al., 2006b] to derive an extremely tight privacy bound. Its central insight is that, in the context of hierarchical decomposition, it is possible to publish a sequence S of 0/1 values using $O(1)$ noise, regardless of the *sensitivity* of S [Dwork et al., 2006b]. In contrast, the standard Laplace mechanism requires that noise amount must be proportional to S's sensitivity.

To demonstrate the applications of PrivTree, we apply it to the private modeling of spatial data, and present a non-trivial extension to tackle sequence data, for which we adopt an advanced Markov model and utilize a sophisticated measure (instead of tuple counts) to decide whether a sub-domain should be split. We experimentally evaluate our algorithms on a variety of real data, and show that they considerably outperform the states of the art in terms of data utility.

In addition, we present an in-depth analysis on the connection between PrivTree and the *sparse vector technique* [Hardt, 2011; Dwork and Roth, 2013; Lee and Clifton, 2014], a technique widely adopted for data publishing under differential privacy. We show that there exists a variant of sparse vector technique [Lee and Clifton, 2014] that could have been used to implement PrivTree, if its privacy guarantees are as claimed in previous work [Lee and Clifton, 2014]. Nevertheless, we prove that the sparse vector technique variant does not satisfy differential privacy, which makes it inapplicable in our context.

In summary, we make the following contributions in this chapter:

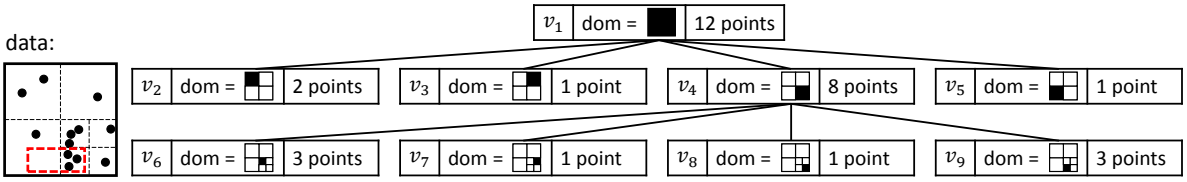


Figure 4.1: An illustration of a spatial decomposition tree

1. We propose PrivTree, a differentially private algorithm for hierarchical decomposition that eliminates the dependency on a pre-defined threshold of the recursion depth.
2. We present applications of PrivTree in modeling spatial and sequence data. (Sections 4.2 and 4.3)
3. We analyze the connection between PrivTree and the sparse vector technique, and point out a misclaim about the latter in [Lee and Clifton, 2014]. (Section 4.4)
4. We conduct extensive experiments to demonstrate the superiority of PrivTree over the states of the art. (Section 4.5)

4.1 Spatial Decompositions

Let D be a set of data points in a multi-dimensional space Ω. A *spatial decomposition* [Samet, 2006; De Berg et al., 2000] of D consists of a tree-structured decomposition of Ω into its sub-domains, along with a partitioning of the data points among the leaves of the decomposition tree. For example, Figure 4.1 illustrates a spatial decomposition of a two-dimensional dataset D that contains 12 data points. The decomposition tree has 9 nodes, namely, $v_1, v_2, \ldots, v_9$, each of which is associated with a sub-domain of Ω (denoted as "dom" and visualized as a black rectangle in Figure 4.1). We refer to each sub-domain as a *region*. The root of the tree, v_1, corresponds to a region that covers the entire Ω; this region is recursively divided into four equal-size sub-regions in the lower levels of the tree, until each leaf node contains a sufficiently small number of data points.

The spatial decomposition in Figure 4.1 is referred to as a *quadtree* [Samet, 2006; De Berg et al., 2000], and is widely adopted in spatial databases for efficient query processing. In particular, suppose that we are to use the quadtree to answer *range count queries*, i.e., queries that ask for the number of data points contained in a rectangle q. In that case, we can pre-compute, for each node v in the quadtree, the number of data points contained in v's region. Then, we can answer any range count query q with a top-down traversal from the root node of the quadtree. Specifically, at the beginning of the traversal, we initialize the query answer as $ans = 0$. After that, for each node v that we traverse, we examine v's region $\mathrm{dom}(v)$, and differentiate four cases:

1. If $\text{dom}(v)$ is disjoint from q, we ignore v;
2. If $\text{dom}(v)$ is fully contained in q, we increase ans by the point count pre-computed for v;
3. If $\text{dom}(v)$ partially intersects q and v is not a leaf node, then we visit every child of v with a region not disjoint from q;
4. If $\text{dom}(v)$ partially intersects q and v is a leaf node, then we inspect the data points in $\text{dom}(v)$, and add to ans the number of points contained in q.

After the traversal terminates, we return ans as the result. For instance, consider a range count query q that corresponds to the dashed-line rectangle in Figure 4.1. To answer q, we only need to examine four nodes, namely, v_1, v_4, v_5, v_9; the other nodes are all ignored since their regions are disjoint from q.

The efficiency of quadtrees results from its adaptiveness to the underlying distribution, i.e., it grows deep into the dense regions of Ω where there are a large number of data points (e.g., the region of v_4 in Figure 4.1), and it ignores those regions that are sparse (e.g., the regions of v_2, v_3, v_5). Such adaptiveness has motivated existing work [Cormode et al., 2012a] to utilize quadtrees for generating private synopses of spatial data. Specifically, the technique in [Cormode et al., 2012a] first applies a differentially private algorithm to generate a quadtree, and then employs the Laplace mechanism to inject noise into the point count of each node. The quadtree and the noisy counts can then be used to answer any range-count query q using the top-down traversal algorithm mentioned above, with two minor modifications. First, whenever we visit a node v whose region is fully contained in q, we add the noisy count associated with v (instead of the exact count) to the query answer ans. Second, if v is a leaf node whose region $\text{dom}(v)$ partially intersects q, then we multiply the noisy count of v by $\frac{|q \cap \text{dom}(v)|}{|\text{dom}(v)|}$ before adding it to ans, where $|\cdot|$ denotes the area of a region. That is, given only the noisy count of v, we estimate the number of data points in $\text{dom}(v)$ that are contained in q, by assuming that the points follow a uniform distribution. The rationale of this approach is that, given the adaptiveness of the quadtree, each leaf node v should cover a region where the data distribution is not highly skewed; otherwise, v should contain a dense sub-region, in which case the quadtree construction algorithm should have further split v (instead of making v a leaf node). This makes it relatively accurate to adopt a uniform assumption when estimating the contribution of v to the answer of q. In Section 4.2, we will present a more detailed analysis of the above quadtree approach, and then use it to motivate our PrivTree algorithm.

Table 4.1: Table of notations

notation	description
n, d	the cardinality and dimensionality of the input dataset D
$\text{Lap}(\lambda)$	a random variable following the Laplace distribution with 0 mean and λ scale
$\text{dom}(v)$	the sub-domain of a node v
$\text{depth}(v)$	the hop distance from a node v to the root of the tree
θ	the threshold used to decide if a node should be split
$c(v), \hat{c}(v)$	the point count of a node v, and its noisy version
$b(v), \hat{b}(v)$	the biased count of a node v, and its noisy version
$\rho(v)$	the privacy risk of a node v (see Equation (4.4))
$\rho^{\top}(v)$	an upper bound of ρ (see Equation (4.6))
β	the fanout of the spatial decomposition tree
δ	the decaying factor used by PrivTree
$\mathcal{I}$	the set of distinct items in a given set D of sequences

Algorithm 4.1: SimpleTree (D, λ, θ, h): returns $\mathcal{T}$

```
initialize a quadtree T with a root node v_1;
set dom(v_1) = Ω, and mark v_1 as unvisited;
while there exists an unvisited node v do
    mark v as visited;
    compute the number c(v) of points in D that are contained in dom(v);
    compute a noisy version of c(v):  ĉ(v) = c(v) + Lap(λ);
    if ĉ(v) > θ and depth(v) < h − 1 then
        split v, and add its children to T;
        mark the children of v as unvisited;
return T;
```

4.2 Private Spatial Decompositions

This section presents our solution for constructing private spatial decompositions. We first revisit the private quadtree approach (in Section 4.1) and discuss its limitations; after that, we elaborate our PrivTree algorithm, analyze its guarantees, and discuss its extensions. Table 4.1 shows the notations that we frequently use.

4.2.1 Private Quadtrees Revisited

Algorithm 4.1 presents a generic version of the private quadtree approach mentioned in Section 4.1. The algorithm takes as input four parameters: (i) a set D of spatial points defined over a multi-dimensional domain Ω, (ii) the scale λ of the Laplace noise to be

used in the construction of the quadtree, (iii) the threshold θ used to decide whether a quadtree node should be split, and (iv) the threshold h on the maximum height of the decomposition tree. The output of the algorithm is a quadtree $\mathcal{T}$ where each node v comes with two pieces of information: the sub-domain of Ω corresponding to v (denoted as $\text{dom}(v)$), and a noisy version of the point count in $\text{dom}(v)$ (denoted as $\hat{c}(v)$). We define the *depth* of v as the hop distance between v and the root of $\mathcal{T}$, and denote it as $\text{depth}(v)$.

The algorithm starts by creating the root note v_1 of $\mathcal{T}$, after which it sets $\text{dom}(v_1) = \Omega$ and marks v_1 as *unvisited* (Lines 1-2). The subsequent part of the algorithm consists of a number of iterations (Lines 3-9). In each iteration, we examine if there is an unvisited node v in $\mathcal{T}$. If such v exists, we mark v as visited, and employ the Laplace mechanism to generate a noisy version $\hat{c}(v)$ of the number of points contained in $\text{dom}(v)$ (Lines 4-6). After that, we split v if the following two conditions simultaneously hold. First, $\hat{c}(v) > \theta$, i.e., $\text{dom}(v)$ is likely to contain a sufficiently large number of points. Second, the height of the tree is smaller than h, which, as we discuss shortly, ensures that the noisy counts generated by the algorithm would not violate differential privacy. If both of the above conditions are met, then we generate v's children and insert them into $\mathcal{T}$ as unvisited nodes (Lines 7-9); otherwise, v becomes a leaf node of $\mathcal{T}$. When all of the nodes in $\mathcal{T}$ become visited, the algorithm terminates and returns $\mathcal{T}$.

Privacy and Utility Analysis. Algorithm 4.1 ensures ε-differential privacy if $\lambda \geq h/\varepsilon$. To understand this, suppose that we insert an arbitrary point t into D. Then, $\mathcal{T}$ has only h nodes whose exact point counts are affected by the insertion of t, i.e., the h nodes whose sub-domains contain t. In addition, the point count of those nodes should change by one after t's insertion. This indicates that the sensitivity (see Definition 2.4) of all point counts in $\mathcal{T}$ equals h, and hence, adding i.i.d. Laplace noise of scale $\lambda \geq h/\varepsilon$ into the counts would achieve ε-differential privacy.

As we mention at the beginning of this chapter, however, requiring $\lambda \geq h/\varepsilon$ makes it rather difficult for Algorithm 4.1 to generate high-quality quadtrees. Specifically, if we set h to a small value, the resulting quadtree $\mathcal{T}$ would not adapt well to the data distribution in D, due to the restriction on the tree height; meanwhile, increasing h would also increase the amount of noise in each $\hat{c}(v)$, which makes Algorithm 4.1 more error-prone in deciding whether a node should be split, thus degrading the quality of $\mathcal{T}$. In other words, the quality of released quadtrees is sensitive to the choice of h. Furthermore, we cannot directly tune h by (i) testing the performance of Algorithm 4.1 on D under different settings of h, and then (ii) selecting the one that yields the best result. The reason is that such a tuning process violates differential privacy: when we change the input data from D to a neighboring dataset D', the tuning process may select a different h, in which case Algorithm 4.1 would use different noise scales for D and D', invalidating its privacy guarantee.

To alleviate the above issue, existing work [Cormode et al., 2012a; Qardaji et al., 2013a,b; Su et al., 2015] resorts to heuristics to choose h without violating differential privacy, and to enhance the performance of Algorithm 4.1, e.g., by avoiding the generation of noisy counts for certain levels of the decomposition tree (so that h can be reduced), and by exploiting correlations among the noisy counts to improve their accuracy [Hay et al., 2010]. However, none of those heuristics is able to thoroughly address the limitations of Algorithm 4.1. As we shown in our experiments in Section 4.5, existing approaches tend to provide inferior data utility, especially when the input data follows a skewed distribution.

4.2.2 Rationale Behind Our Solution

To remedy the deficiency of Algorithm 4.1, we aim to eliminate the requirement that $\lambda \geq h/\varepsilon$, and make λ a constant instead. This would not only resolve the dilemma in choosing h, but also unleash the potential of quadtrees as the tree height is no longer restricted. Towards this end, we first make a simple observation: after we finish constructing the quadtree $\mathcal{T}$ in Algorithm 4.1, we could remove all noisy counts associated with the intermediate nodes, and release only the noisy counts for the leaf nodes as well as the sub-domains of all nodes. The released tree, denoted as $\mathcal{T}'$, could still be used for query processing, since we can re-generate an alternative count for each intermediate node v in $\mathcal{T}'$ by summing up the published noisy counts of the leaf nodes under v. Intuitively, $\mathcal{T}'$ reveals less information than $\mathcal{T}$ does, and hence, we might use less noise in $\mathcal{T}'$ to achieve ε-differential privacy.

However, the above intuition does not hold in general, as $\mathcal{T}'$ and $\mathcal{T}$ require the same amount of noise to enforce the same privacy guarantee. To explain, consider two neighboring datasets D and D', such that D is obtained by inserting a point t into D'. Let $v_1, v_2, \ldots, v_h$ be the h nodes in $\mathcal{T}'$ whose sub-domains contain t. Then, these h nodes should form a path from the root of $\mathcal{T}$ to a leaf node. Furthermore, for any v_i, the point count of v_i is decreased by one when we change the input dataset from D to D'. We use $c(v_i)$ to denote v_i's exact point count on D.

Without loss of generality, assume that v_h is a leaf node (i.e., $v_1, \ldots, v_{h-1}$ are all intermediate nodes). Let $\Pr[D \rightarrow \mathcal{T}']$ (resp. $\Pr[D' \rightarrow \mathcal{T}']$) denote the probability that we obtain $\mathcal{T}'$ from D (resp. D') given fixed λ, θ, and h. Then,

$$\ln\left(\frac{\Pr[D \rightarrow \mathcal{T}']}{\Pr[D' \rightarrow \mathcal{T}']}\right) = \sum_{i=1}^{h-1} \ln\left(\frac{\Pr[c(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[c(v_i) - 1 + \mathrm{Lap}(\lambda) > \theta]}\right) + \ln\left(\frac{\Pr[c(v_h) + \mathrm{Lap}(\lambda) = \hat{c}(v_h)]}{\Pr[c(v_h) - 1 + \mathrm{Lap}(\lambda) = \hat{c}(v_h)]}\right).$$

By Equation (2.2), for any $c(v_h) < \hat{c}(v_h)$,

$$\ln\left(\frac{\Pr[c(v_h) + \mathrm{Lap}(\lambda) = \hat{c}(v_h)]}{\Pr[c(v_h) - 1 + \mathrm{Lap}(\lambda) = \hat{c}(v_h)]}\right) = \frac{1}{\lambda}. \tag{4.1}$$

In addition, for any v_i ($i \in [1, h-1]$) with $c(v_i) \leq \theta$,

$$\ln\left(\frac{\Pr[c(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[c(v_i) - 1 + \mathrm{Lap}(\lambda) > \theta]}\right) = \frac{1}{\lambda}. \tag{4.2}$$

Therefore, when $c(v_i) \leq \theta$ holds for every v_i ($i \in [1, h-1]$),

$$\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = \frac{h}{\lambda}. \tag{4.3}$$

By Definition 2.2, this indicates that λ must be at least h/ε to ensure that $\mathcal{T}'$ achieves ε-differential privacy.

In summary, $\mathcal{T}'$ is no better than $\mathcal{T}$ in terms of the amount of noise required, because of the negative result in Equations (4.1) and (4.2). In other words, *in the worst case*, releasing the boolean result of $c(v_i) + \mathrm{Lap}(\lambda) > \theta$ incurs the same privacy cost as releasing $c(v_i) + \mathrm{Lap}(\lambda)$ directly. That said, if $c(v_i) > \theta$ for some v_i, then Equation (4.2) does not hold, in which case $\mathcal{T}'$ could entail a smaller privacy cost than $\mathcal{T}$ does. To illustrate this, we denote the l.h.s. of Equation (4.2) as a function ρ of $c(v_i)$, i.e.,

$$\rho(x) = \ln\left(\frac{\Pr[x + \mathrm{Lap}(\lambda) > \theta]}{\Pr[x - 1 + \mathrm{Lap}(\lambda) > \theta]}\right), \tag{4.4}$$

and we plot ρ in Figure 4.2. (Note that the y-axis of Figure 4.2 is in a logarithmic scale.) Observe that, when $x = c(v) \geq \theta + 1$, $\rho(x)$ decreases exponentially with the increase of x. This indicates that $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right)$ could be much smaller than h/λ, if $c(v_i) \geq \theta + 1$ holds for all $i \in [1, h-1]$. For example, if $c_{h-1} \geq \theta + 1$ and $c(v_i) - c(v_{i+1})$ is at least a constant for all $i \in [1, h-2]$, then

$$\sum_{i=1}^{h-1} \rho\Big(c(v_i)\Big) = \Theta\left(\frac{1}{\lambda}\right), \tag{4.5}$$

due to the exponential decrease of $\rho(v_i)$. In that case, we have $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = \Theta\left(1/\lambda\right)$ instead of $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = h/\lambda$, which would enable us to set λ as a constant independent of h.

The above analysis leads to an interesting question: can we ensure that Equation (4.5) holds for any input dataset? In Section 4.2.3, we will give an affirmative answer to this question. The basic idea of our method is to add a bias term to each $c(v_i)$, so that

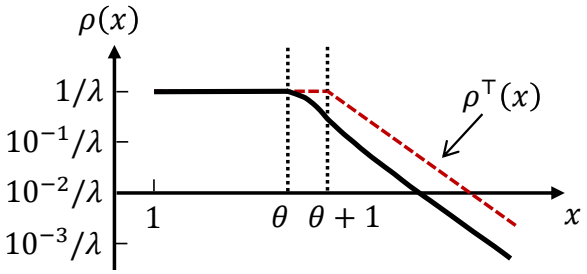


Figure 4.2: An illustration of $\rho(x)$ and $\rho^{\top}(x)$

$c(v_i) - c(v_{i+1})$ $(i \in [1, h-2])$ is larger than a constant of choice. In addition, the bias term is independent of the input data, which guarantees that its usage does not leak any private information. The derivation of the bias term requires a careful analysis of $\rho(x)$. To simplify our analysis, we devise a simple upper bound of $\rho(x)$:

Lemma 4.1 *Let $\rho^{\top}$ be a function such that*

$$\rho^{\top}(x) = \begin{cases} 1/\lambda, & \text{if } x < \theta + 1; \\ \frac{1}{\lambda} \exp\left(\frac{\theta+1-x}{\lambda}\right), & \text{otherwise.} \end{cases} \tag{4.6}$$

Then, $\rho(x) \leq \rho^{\top}(x)$ for any x.

Figure 4.2 shows $\rho^{\top}$ with a dashed line. Observe that it closely captures the exponential decrease of ρ when $x \geq \theta + 1$.

4.2.3 The PrivTree Algorithm

Algorithm 4.2 presents our PrivTree technique for private spatial decomposition. As with Algorithm 4.1, PrivTree asks for a spatial dataset D, the scale λ of Laplace noise to be used, and a threshold θ for deciding whether a node should be split. However, it does not request a threshold h on the maximum tree height; instead, it requires a positive number δ, the usage of which will be clarified shortly. The output of PrivTree is a quadtree $\mathcal{T}$, with the point count associated with each node removed. That is, $\mathcal{T}$ reveals the sub-domain of each node v, but conceals all information about $c(v)$. In Section 4.2.4, we will explain how we obtain the point count of each node, as well as our choices of θ and δ.

In a nutshell, PrivTree is similar to Algorithm 4.1 in that it also (i) generates $\mathcal{T}$ by recursively splitting a root node v_1 whose region $\text{dom}(v_1)$ covers the whole data space Ω, and (ii) decides whether a node v should be split based on a noisy point count of v. However, the method for obtaining noisy counts marks the crucial difference between the two algorithms. Specifically, given a node v, PrivTree does not generate its noisy

Algorithm 4.2: PrivTree (D, λ, θ, δ): returns $\mathcal{T}$

initialize a quadtree $\mathcal{T}$ with a root node v_1;
set $\mathrm{dom}(v_1) = \Omega$, and mark v_1 as unvisited;
while *there exists an unvisited node* v **do**
    mark v as visited;
    compute a biased point count for v with decaying factor δ: $b(v) = c(v) - \mathrm{depth}(v) \cdot \delta$;
    adjust $b(v)$ if it is excessively small: $b(v) = \max\{b(v), \theta - \delta\}$;
    compute a noisy version of $b(v)$: $\hat{b}(v) = b(v) + \mathrm{Lap}(\lambda)$;
    if $\hat{b}(v) > \theta$ **then**
        split v, and add its children to $\mathcal{T}$;
        mark the children of v as unvisited;
return $\mathcal{T}$ with all point counts removed;

count by directly adding Laplace noise to $c(v)$. Instead, PrivTree first computes a biased count $b(v) = c(v) - \mathrm{depth}(v) \cdot \delta$, and checks if it is smaller than $\theta - \delta$; if it is, then PrivTree increases it to $\theta - \delta$. In other words,

$$b(v) = \max\Big\{\theta - \delta,\ \ c(v) - \mathrm{depth}(v) \cdot \delta\Big\}. \tag{4.7}$$

After that, PrivTree produces a noisy count $\hat{b}(v) = b(v) + \mathrm{Lap}(\lambda)$, and splits v if $\hat{b}(v)$ is larger than the given threshold θ. Notice that PrivTree does not restrict the height of $\mathcal{T}$, as the decision to split any node v solely depends on $\hat{b}(v)$.

Privacy Analysis. Consider any quadtree $\mathcal{T}$ output by PrivTree, and any two neighboring datasets D and D', such that D is obtained by inserting a point t into D'. In what follows, we show that setting $\lambda = \Theta(1/\varepsilon)$ is sufficient for ε-differential privacy, i.e.,

$$-\varepsilon \leq \ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) \leq \varepsilon. \tag{4.8}$$

The proof for the first inequality in Equation (4.8) is relatively straightforward. For any node v in $\mathcal{T}$, let $c(v)$ be v's point count on D, and $b(v)$ be the biased version of $c(v)$ generated from Equation (4.7). Let $c'(v)$ and $b'(v)$ be the counterparts of $c(v)$ and $b(v)$, respectively, given D' as the input. Then, we have $c(v) = c'(v)$ for all nodes v in $\mathcal{T}$, except for the nodes whose sub-domains contain t. Note that those nodes should form a path from the root of $\mathcal{T}$ to a leaf. Let k be the length of the path, and v_i be i-th node in the path, with v_1 denoting the root of $\mathcal{T}$. We have $c'(v_i) = c(v_i) - 1$ and

$$b'(v_i) = b(v_i) - 1, \text{if } b(v_i) \geq \theta - \delta + 1. \tag{4.9}$$

Then, by Equation (2.2),

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) = \sum_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[b'(v_i) + \mathrm{Lap}(\lambda) > \theta]}\right) + \ln\left(\frac{\Pr[b(v_k) + \mathrm{Lap}(\lambda) \le \theta]}{\Pr[b'(v_k) + \mathrm{Lap}(\lambda) \le \theta]}\right)$$
$$\ge 0 - \frac{1}{\lambda} = -\frac{1}{\lambda}.$$

This indicates that $\lambda \ge 1/\varepsilon$ ensures the first inequality in Equation (4.8).

Next, we prove the second inequality in Equation (4.8) by analyzing $\ln\left(\frac{\Pr[b(v_i)+\mathrm{Lap}(\lambda)>\theta]}{\Pr[b'(v_i)+\mathrm{Lap}(\lambda)>\theta]}\right)$, which we refer to as the *privacy cost* of v_i. The high-level idea of our proof is as follows. First, due to the way that we generate biased counts, each node v_i's bias count $b(v_i)$ is at least a constant δ smaller than that of its parent v_{i-1}, as long as $b(v_i) \ge \theta + 1$. Based on this observation and Lemma 4.1, we show that all nodes v_i with $b(v_i) \ge \theta + 1$ incur a total privacy cost of $\Theta(1/\varepsilon)$. After that, we prove that the total privacy cost of the remaining nodes is also $\Theta(1/\varepsilon)$.

By the definition of $v_1, \ldots, v_k$, we have $c(v_i) \ge c(v_{i+1})$ and $\mathrm{depth}(v_i) = \mathrm{depth}(v_{i+1}) - 1$ for any $i \in [1, k-1]$. This indicates that $b(v_i) \ge b(v_{i+1}) \ge \theta - \delta$, due to Equation (4.7). Without loss of generality, assume that there exists $m \in [1, k-1]$, such that $b(v_m) \ge \theta - \delta + 1$ and $b(v_{m+1}) = \theta - \delta$. Then,

$$\begin{cases} b(v_{i-1}) \ge b(v_i) + \delta \ge \theta + 1, & \text{if } i \in [2, m]; \\ b(v_i) = \theta - \delta, & \text{otherwise.} \end{cases} \tag{4.10}$$

Combining Equations (4.9) and (4.10), we have $b'(v_i) = b(v_i)$ when $i > m$, and $b'(v_i) = b(v_i) - 1$ otherwise. Therefore,

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) = \sum\nolimits_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[b'(v_i) + \mathrm{Lap}(\lambda) > \theta]}\right) + \ln\left(\frac{\Pr[b(v_k) + \mathrm{Lap}(\lambda) \le \theta]}{\Pr[b'(v_k) + \mathrm{Lap}(\lambda) \le \theta]}\right)$$
$$\le \sum\nolimits_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[b'(v_i) + \mathrm{Lap}(\lambda) > \theta]}\right)$$
$$= \sum\nolimits_{i=1}^{m} \ln\left(\frac{\Pr[b(v_i) + \mathrm{Lap}(\lambda) > \theta]}{\Pr[b(v_i) - 1 + \mathrm{Lap}(\lambda) > \theta]}\right) = \sum\nolimits_{i=1}^{m} \rho\Big(b(v_i)\Big),$$

where $\rho(\cdot)$ is as defined in Equation (4.4). By Lemma 4.1 and Equation (4.10),

$$\sum\nolimits_{i=1}^{m} \rho\Big(b(v_i)\Big) \le \sum\nolimits_{i=1}^{m} \rho^{\top}\Big(b(v_i)\Big) = \rho^{\top}\Big(b(v_m)\Big) + \sum\nolimits_{i=1}^{m-1} \frac{1}{\lambda} \exp\left(\frac{\theta + 1 - b(v_i)}{\lambda}\right)$$
$$\le \frac{1}{\lambda} + \frac{1}{\lambda} \cdot \frac{1}{1 - \exp(-\delta/\lambda)} = \frac{1}{\lambda} \cdot \frac{2e^{\delta/\lambda} - 1}{e^{\delta/\lambda} - 1}.$$

Therefore, if we set $\delta = \gamma \cdot \lambda$, where γ is a constant, then

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) \;=\; \sum_{i=1}^{m} \rho\Big(b(v_i)\Big) \;\leq\; \frac{1}{\lambda} \cdot \frac{2e^{\gamma}-1}{e^{\gamma}-1} \;=\; \Theta\left(\frac{1}{\lambda}\right).$$

Summing up the above analysis, we have the following theorem:

Theorem 4.1 PrivTree *satisfies ε-differential privacy if $\lambda \geq \frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ and $\delta = \gamma \cdot \lambda$ for some $\gamma > 0$.*

4.2.4 Noisy Counts and Parameterization

Generation of Noisy Counts. Recall that PrivTree outputs a quadtree with the point count for each node removed. However, if a quadtree with noisy counts is needed, we can easily obtain it by adding an extra step to PrivTree. In particular, given a dataset D, we first invoke PrivTree to produce an $\varepsilon/2$-differentially private quadtree $\mathcal{T}$. After that, for each leaf node v of $\mathcal{T}$, we publish a noisy version of v's point count using Laplace noise of scale $2/\varepsilon$. It can be verified that this step satisfies $\varepsilon/2$-differential privacy. Then, by the composition rule in Lemma 2.1, the generation of $\mathcal{T}$ and the noisy counts as a whole achieves ε-differential privacy. Finally, we compute a noisy count for each intermediate node v in $\mathcal{T}$, by taking the sum of the noisy counts of all leaf nodes under v.

Choice of δ. As shown in Theorem 4.1, when $\delta = \gamma \cdot \lambda$, PrivTree needs to use a noise scale $\lambda \geq \frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ to achieve ε-differential privacy. Intuitively, the choice of δ is a balancing act between the amount of bias and the amount of noise in each biased noisy count $\hat{b}(v)$ used by PrivTree. In particular, if δ is small with respect to λ, then the bias term in each $\hat{b}(v)$ is small, but the noise amount in $\hat{b}(v)$ would need to be large, since $\frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ increases when $\gamma = \delta/\lambda$ decreases. In contrast, if δ is large with respect to λ, then each $\hat{b}(v)$ would have small noise but a large bias.

That said, we observe that there is a more important factor in choosing δ. To explain, consider a node v with a biased count $b(v) = \theta - \delta$. Ideally, we would like PrivTree to avoid splitting such a node v, as its point count is likely to be small. Nevertheless, if $b(v)+\text{Lap}(\lambda) > \theta$, then PrivTree would split v and insert its children into the quadtree. In turn, each of v's children also has a certain probability to be split, and so on. If δ is excessively small, then each offspring of v has a relatively large splitting probability, in which case the splitting process may not *converge*, i.e., PrivTree may keep generating offsprings of v and does not terminate.

To address the above issue, we set δ in such a way to ensure that if $b(v) = \theta - \delta$, then in expectation, only 2 nodes would be generated from the subtree under v (including v

itself). Specifically, we set $\delta = \lambda \cdot \ln \beta$, where β denote the *fanout* of $\mathcal{T}$, i.e., the number of children that each intermediate node in $\mathcal{T}$ has. (For example, $\beta = 4$ if $\mathcal{T}$ is a two-dimensional quadtree.) By Equation (2.2), this setting of δ guarantees that any node v with $b(v) = \theta - \delta$ has $\frac{1}{2\beta}$ probability to be split. Formally, we have the following lemma:

Lemma 4.2 *Let $\mathcal{T}$ be the output of* PrivTree *given $\delta = \lambda \cdot \ln \beta$, and $\mathcal{T}^*$ be the output of* PrivTree *when it sets $\hat{b}(v) = c(v)$ for each node v (i.e., no noise or bias is introduced in the split decisions). Then, $\mathbb{E}[|\mathcal{T}|] \leq 2 \cdot |\mathcal{T}^*|$ whenever $|T^*| > 1$, where $|\cdot|$ denotes the number of nodes in a tree.*

The above setting of δ leads to the following corollary:

Corollary 4.1 PrivTree *satisfies ε-differential privacy if $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{1}{\varepsilon}$ and $\delta = \lambda \cdot \ln \beta$, where β is the fanout of $\mathcal{T}$.*

Choice of θ. Intuitively, the threshold θ serves the purpose of ensuring that each leaf node v of $\mathcal{T}$ contains a sufficiently large point count $c(v)$, so that if we choose to output a noisy version of $c(v)$, it would not overwhelmed by the Laplace noise injected. The choice of θ, however, is complicated by the fact that PrivTree adds a negative bias to the point count of a node when it decides whether or not to split the node. In particular, due to the negative bias, even $\theta = 0$ could ensure that a node v with $b(v) > \theta$ has a sufficient large point count. Therefore, we use $\theta = 0$ in our implementation of PrivTree, and we observe that it leads to reasonably good results in our experiments.

4.2.5 Extensions

Although we have presented PrivTree in the context of spatial decomposition, we note that it could be extended in several different aspects for other applications. First, the decomposition tree used by PrivTree does not have to be a quadtree, but can be any other tree structure instead. For example, suppose that we are given a multi-dimensional dataset D containing both numeric and categorical attributes, and that each categorical attribute has a taxonomy. Then, we can still apply PrivTree on D to generate a private synopsis of D, by splitting each numeric dimension of D according to a binary tree and each categorical dimension based on its taxonomy.

Second, when PrivTree decides whether or not a node v should be split, the decision does not have to be based on the count of tuples contained in $\mathrm{dom}(v)$, but can also be based on any other score function $\mu(v)$ that is *monotonic*, i.e., $\mu(v) \leq \mu(u)$ whenever v is a child node of another node u. The rationale is that, as long as μ is monotonic, we can add a bias to the score of each node v to ensure that it is at least a constant smaller that the score of v's parent. Then, we can apply Lemma 4.1 to show that PrivTree

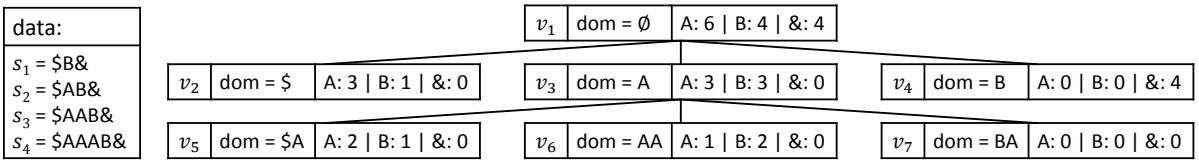


Figure 4.3: An illustration of a prediction suffix tree

guarantees differential privacy, given that λ is properly set based on the sensitivity of the score function μ. In Section 4.3, we will apply this idea to extend PrivTree for private modeling of sequence data.

Finally, although the privacy analysis of PrivTree (in Section 4.2.3) assumes that the presence or absence of a tuple t only affects one leaf node and its ancestors, it can be extended to the case when multiples leaf nodes and their ancestors are impacted. In particular, if at most x leaf nodes can be affected, then we can apply PrivTree with the noise scale λ enlarged x times. The intuition is that each affected leaf node, along its ancestors, incurs one unit of privacy cost, which in turn requires one unit of noise to mitigate; as such, when there are x affected leaf nodes, we need x units of noise for sufficient privacy protection.

4.3 Private Markov Models

This section presents an extension of PrivTree for constructing Markov models on sequence data. We first introduce the basic concepts of sequences and Markov models in Section 4.3.1. After that, we elaborate our PrivTree extension in Section 4.3.2, and compare it with existing solutions in Section 4.3.3.

4.3.1 Sequence Data and Markov Models

Given a finite alphabet $\mathcal{I}$, a *sequence* s of length l over $\mathcal{I}$ is an ordered list $x_1x_2\cdots x_l$, where each x_i ($i \in [1, l]$) is a symbol in $\mathcal{I}$. For convenience, we abuse notation and write $s = \$x_1x_2\cdots x_l\&$, where \$ and & are two special symbols that mark the beginning and the end of a sequence, respectively. Sequences are frequently used to represent user behavioral data, such as trajectories, web navigation traces, and product purchasing histories.

Markov models are a type of stochastic models commonly used to characterize sequence data. They assume the *Markov property* [Ron et al., 1996], i.e., a symbol x in a sequence s is decided by a few symbols that immediately proceeds x in s, but not any others. A Markov model over D is often represented as *prediction suffix tree* [Ron et al., 1996; Begleiter et al., 2004], where each node v is associated with a *predictor*

string $\mathrm{dom}(v)$, as well as a *prediction histogram* $\mathrm{hist}(v)$. In particular, $\mathrm{dom}(v)$ consists of symbols in $\mathcal{I} \cup \{\$\}$, while $\mathrm{hist}(v)$ contains a count for each symbol x in $\mathcal{I} \cup \{\&\}$. The count, denoted as $\mathrm{hist}(v)[x]$, is computed as follows. We first inspect all occurrences of $\mathrm{dom}(v)$ in the sequences in D, and count the number y of occurrences when $\mathrm{dom}(v)$ is immediately followed by the symbol x; after that, we set $\mathrm{hist}(v)[x] = y$. In other words, $\mathrm{hist}(v)[x]$ indicates how often an appearance of $\mathrm{dom}(v)$ would immediately lead to an appearance of x in a sequence.

For example, Figure 4.3 illustrates a prediction suffix tree constructed over a set D containing four sequences s_1, s_2, s_3, s_4, over an alphabet $\mathcal{I} = \{A, B\}$. The node v_6 has a predictor string $\mathrm{dom}(v_6) = AA$, and prediction histogram $\mathrm{hist}(v_6)$ that contains a count for each element in $\{A, B, \&\}$. The counts in the histogram sum up to 3, since the string AA appears 3 times in D, i.e., once in s_3 and twice in s_4. In addition, $\mathrm{hist}(v_6)[A] = 1$ because, among the 3 occurrences of AA, only one is immediately followed by A (i.e., the first occurrence of AA in s_4). The root node v_1 has an empty predictor string $\mathrm{dom}(v_1) = \emptyset$, and its prediction histogram counts the occurrences of each individual symbol in $\mathcal{I} \cup \{\&\}$, e.g., $\mathrm{hist}(v_1)[A] = 6$ since the symbol A appears 6 times in total in D.

The nodes in a prediction suffix tree are organized in such a way that each node v has $|\mathcal{I}| + 1$ children. Furthermore, for each child v' of v, $\mathrm{dom}(v')$ is obtained by adding a symbol in $\mathcal{I} \cup \{\$\}$ in the beginning of $\mathrm{dom}(v)$. That is, $\mathrm{dom}(v)$ is a suffix of $\mathrm{dom}(v')$. For example, in the prediction suffix tree in Figure 4.3, v_3 is a parent of v_5; accordingly, $\mathrm{dom}(v_5) = \$A$ is obtained by adding the symbol \$ to the beginning of $\mathrm{dom}(v_3) = A$. The intuition here is that (i) each node v in a prediction suffix tree provides a way to predict the "next symbol" in a sequence based on a "predicate" $\mathrm{dom}(v)$, and (ii) when we split v, each child node would have a longer "predicate" that provides a more specific predication.

A prediction suffix tree $\mathcal{T}$ can be used to support a wide range of queries, such as estimating the number of times that a query string s_q appears in the sequences in D. Specifically, given $s_q = x_1 x_2 \ldots x_l$, we first inspect the root node v_1's prediction histogram $\mathrm{hist}(v_1)$, and then initialize a temporary answer $ans = \mathrm{hist}(v_1)[x_1]$. After that, we examine x_i ($i \in [2, l]$) in ascending order of i. For each x_i, we consider the length-$(i-1)$ prefix of s_q, i.e., $s_i^* = x_1 x_2 \ldots x_{i-1}$. We identify the node v in $\mathcal{T}$ whose predictor string is the longest suffix of s_i^*. Then, we compute the sum of the counts in v's prediction histogram $\mathrm{hist}(v)$, referred to as the *magnitude* of the histogram and denoted as $\|\mathrm{hist}(v)\|_1$. After that, we set

$$ans = ans \cdot \frac{\mathrm{hist}(v)[x_i]}{\|\mathrm{hist}(v)\|_1}, \tag{4.11}$$

i.e., we multiple ans by the probability that the "next symbol" equals x_i, as predicted by $\mathrm{hist}(v)$. When all x_i ($i \in [1, l]$) are examined, we return ans as the query answer.

For example, consider a query sequence $s_q = AB$ on the prediction suffix tree in Figure 4.3. We first visit the root node v_1, and initialize $ans = \text{hist}(v_1)[A] = 6$. After that, we consider the length-1 prefix of s_q, i.e., $s_2^* = A$. We identify v_3 as the node whose predictor string is the longest suffix of s_2^*, and we set $ans = ans \cdot \frac{\text{hist}(v_3)[B]}{\|\text{hist}(v_3)\|_1} = 3$. Finally, we return $ans = 3$ as the answer.

In addition to the aforementioned query type, we can also utilize a prediction suffix tree $\mathcal{T}$ to generate a *synthetic sequence dataset*, by sampling sequences from $\mathcal{T}$ one by one. Specifically, to generate a sequence, we start from an initial sequence $s_0 = \$$ and insert symbols into s_0 iteratively. In the i-th iteration ($i \geq 1$), we inspect the sequence s_{i-1}, and identify the node v in $\mathcal{T}$ whose predictor string is the longest suffix of s_{i-1}. Then, we sample a symbol x_i from the symbol distribution represented by $\text{hist}(v)$, i.e., $\Pr[x_i = x] = \frac{\text{hist}(v)[x]}{\|\text{hist}(v)\|_1}$. After that, we insert x_i to the end of s_{i-1}, and denote the resulting sequence as s_i. If x_i happens to be &, then we return s_i as the result.

4.3.2 Extension of PrivTree

To construct a prediction suffix tree $\mathcal{T}$ on a sequence dataset D, we can start from a root node v_1 with a predictor string $\text{dom}(v_1) = \emptyset$, and then recursively split v_1. This motivates us to adopt PrivTree for the generation of differentially private prediction suffix trees. However, we can no longer use a node v's noisy count $c(v)$ to decide whether v should be split, since $c(v)$ is undefined on a prediction suffix tree. Instead, as discussed in Section 4.2.5, we can redefine $c(v)$ as a score function that measures the suitability of v for splitting. In the non-private setting, existing work [Ron et al., 1996] typically avoids splitting a node v if any of the following conditions is satisfied:

C1. $\text{dom}(v)$ *starts with* \$. In this case, no more symbol can be added to the beginning of $\text{dom}(v)$; thus, v cannot be split.

C2. *The magnitude of* $\text{hist}(v)$ *is small.* The rationale is that, when $\|\text{hist}(v)\|_1$ is small, further splitting v results in child nodes v' whose prediction histograms $\text{hist}(v')$ have even smaller magnitudes. In that case, the symbol distribution captured by $\text{hist}(v')$ is obtained from a tiny sample set of sequences, which leads to poor prediction accuracy.

C3. *The entropy*[1] *of* $\text{hist}(v)$ *is small.* This is because when $\text{hist}(v)$ has a small entropy, there is little uncertainty in the symbol prediction given by $\text{hist}(v)$; as such, there is little benefit in splitting v. (See v_4 in Figure 4.3 for an example.)

Suppose that we are to adopt the above conditions into PrivTree. Condition C1 can be straightforwardly applied, since it only depends on $\text{dom}(v)$ and does not rely on D, i.e., it does not leak private information. In contrast, conditions C2 and C3 cannot be directly adopted since the counts in $\text{hist}(v)$ depend on D. To address this issue, we aim to design a score function $c(\cdot)$ for PrivTree with the following two properties:

[1]Here we treat $\text{hist}(v)$ as a probability distribution.

Algorithm 4.3: PrivTree (D, λ, θ, δ): returns $\mathcal{T}$

```
1  initialize a prediction suffix tree T with a root node v_1;
2  set dom(v_1) = ∅ and mark v_1 as unvisited;
3  while there exists an unvisited node v do
4      mark v as visited;
5      if dom(v) starts with $ then continue;
6      compute hist(v), then a biased score for v with decaying factor δ:
         b(v) = c(v) − depth(v) · δ;
7      adjust b(v) if it is excessively small: b(v) = max{b(v), θ − δ};
8      compute a noisy version of b(v):   b̂(v) = b(v) + Lap(λ);
9      if b̂(v) > θ then
10         split v, and add its children to T;
11         mark the children of v as unvisited;
12 return T with all node scores and histograms removed;
```

P1. $c(\cdot)$ is monotonic, i.e., $c(v) \leq c(u)$ for any node v and its parent u. This, as discussed in Section 4.2.5, is required to ensure that PrivTree satisfies differential privacy.

P2. If a node v's prediction histogram has a small magnitude or a small entropy, then $c(v)$ tends to be small. This is motivated by conditions C2 and C3 mentioned above.

Our construction of $c(\cdot)$ is based on the following observation: if a prediction histogram has a small entropy, it often has one symbol count that dominates the others, because a small entropy implies that the distribution of symbols in the histogram is skewed. (v_4 in Figure 4.3 shows an example.) Motivated by this, we define $c(v)$ as

$$c(v) = \| \operatorname{hist}(v) \|_1 - \max_{x \in \mathcal{I} \cup \{\&\}} \operatorname{hist}(v)[x], \tag{4.12}$$

i.e., $c(v)$ equals the magnitude of $\operatorname{hist}(v)$ minus the largest count in $\operatorname{hist}(v)$. The intuition is that if the magnitude of $\operatorname{hist}(v)$ is small, then $c(v)$ must be small, regardless of the largest count in $\operatorname{hist}(v)$; on the other hand, if the entropy of $\operatorname{hist}(v)$ is small, then the largest count in $\operatorname{hist}(v)$ tends to be close to the magnitude of $\operatorname{hist}(v)$ (since the count often dominates all other counts in $\operatorname{hist}(v)$), which results in a small $c(v)$ as well. Thus, $c(v)$ fulfills property P2. The following lemma show that $c(v)$ also satisfies property P1.

Lemma 4.3 *$c(\cdot)$ is a monotonic function.*

In summary, we can construct a private prediction suffix tree on a sequence dataset D, using PrivTree with four minor changes (see Algorithm 4.3). First, in Lines 1-2 of Algorithm 4.3, $\mathcal{T}$ is a prediction suffix tree with a fanout $|\mathcal{I}| + 1$ (instead of a quadtree),

and v_1 is the root with a predictor string $\text{dom}(v_1) = \emptyset$. Second, in Line 5, we skip the node whose predictor string starts with \$, due to condition C1. Third, in Line 6, $c(v)$ is as defined in Equation (4.12). Last, in Line 12, we return $\mathcal{T}$ after removing the biased score $\hat{b}(v)$ and the prediction histogram $\text{hist}(v)$ of each node v.

Privacy Analysis. To analyze the privacy guarantee of the modified PrivTree, we first introduce an assumption that is also adopted in prior work [Chen et al., 2012a] on sequence data publication under differential privacy: we assume that the length of each sequence in D, when taking into account & but not \$, is at most $l^\top$, where $l^\top$ is a known constant. To explain why this assumption is needed, consider that we insert an *infinite* sequence s into D to obtain a neighboring dataset D'. In that case, the insertion of s incurs unbounded changes in the histogram counts of the prediction suffix tree, which makes it impossible to achieve differential privacy. In practice, if $l^\top$ is unknown, we can choose an appropriate $l^\top$ in a differentially private manner[2], and truncate any sequence that is excessively long. Specifically, if $s = \$x_1x_2 \ldots x_{l^\top}\&$, then we truncate it to $s = \$x_1x_2 \ldots x_{l^\top}$, i.e., s becomes an open-ended sequence. Note that the removal of & from s does not affect the construction of the prediction suffix tree.

Combining this assumption on $l^\top$ with Corollary 4.1, we prove the privacy guarantee of the modified PrivTree as follows.

Theorem 4.2 *Let $\beta = |\mathcal{I}| + 1$. The modified* PrivTree *ensures ε-differential privacy when $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{l^\top}{\varepsilon}$ and $\delta = \lambda \cdot \ln \beta$.*

Generation of Noisy Prediction Histograms. As prediction histograms are removed from the output $\mathcal{T}$ of PrivTree, recovering them consumes extra privacy budget. Specifically, for each leaf node v in $\mathcal{T}$, we materialize its prediction histogram $\text{hist}(v)$ from D, and then release a noisy version, denoted as $\widehat{\text{hist}}(v)$, by adding Laplace noise $\text{Lap}(\lambda)$ into each histogram count. This step also achieves ε-differential privacy.

Theorem 4.3 *Let $\mathcal{T}$ be a prediction suffix tree. Releasing noisy prediction histograms of $\mathcal{T}$'s leaf nodes by adding Laplace noise* $\text{Lap}(\lambda)$ *into each histogram count achieves ε-differential privacy, if $\lambda \geq \frac{l^\top}{\varepsilon}$.*

From noisy histograms of leaf nodes, one can easily derive other histograms in $\mathcal{T}$, with no extra access to the original database. For each non-leaf node u, we construct histogram $\widehat{\text{hist}}(u)$ by aggregating the histograms of its child nodes, such that for any symbol $x \in \mathcal{I} \cup \{\&\}$,

$$\widehat{\text{hist}}(u)[x] = \sum_{v \text{ is a child of } u} \widehat{\text{hist}}(v)[x].$$

[2]Such $l^\top$ can be chosen by first identifying the 90% or 95% quantile of the sequence lengths in D, and then computing a differentially private version of the quantile [Smith, 2011].

Finally, if any noisy histogram in $\mathcal{T}$ has a negative count, we reset the count to zero. This is to ensure that each histogram represents a distribution of symbols.

Last, we clarify some parameter settings. First, we set the threshold θ in Algorithm 4.3 to 0, just following our analysis in Section 4.2.4. Second, we divide the privacy budget ε between PrivTree and the generation of noisy prediction histograms, such that the former achieves $\frac{\varepsilon}{\beta}$-differential privacy and the latter achieves $\frac{\varepsilon(\beta-1)}{\beta}$-differential privacy. To explain, recall that in PrivTree, we inject Laplace noise into each node v's score $c(v)$, which equals the sum of $\beta - 1$ counts in v's prediction histogram hist(v) (i.e., all counts except the largest one). Meanwhile, when constructing noisy histograms, we directly add Laplace noise to each count y in the prediction histograms. Intuitively, $c(v)$ is roughly $\beta-1$ times more resilient to noise than y. Therefore, we set the privacy budget for histogram generation to be $\beta - 1$ times the budget for PrivTree, so as to balance the relative accuracy of $c(v)$ and y after noise injection.

4.3.3 Comparison with Previous Work

There exist two differentially private methods [Chen et al., 2012a,b] for modeling sequence data, and they both utilize hierarchical decompositions for model construction. However, they considerably differ from PrivTree in three aspects. First, they model sequences based on their prefixes [Chen et al., 2012b] or n-grams [Chen et al., 2012a], while PrivTree is based on a prediction suffix tree representation of the *variable length Markov chain model* [Ron et al., 1996]. Second, their algorithms for hierarchical decompositions are similar in spirit to Algorithm 4.1, due to which they also require a pre-defined threshold h on the maximum height of the decomposition tree. Consequently, they suffer from similar deficiencies to those of Algorithm 4.1, i.e., they cannot generate accurate models because of the dependency on h. Third, when constructing a decomposition tree, the methods in [Chen et al., 2012b,a] decide whether a node be split based only on a count associated with the node, whereas PrivTree adopts a more advanced strategy that takes into account three conditions commonly considered in the non-private setting. The above differences make PrivTree an effective approach for modeling sequence data, as we demonstrate in our experiments in Section 4.5.

4.4 Connections to the Sparse Vector Technique

In this section, we investigate the connection between PrivTree and the *sparse vector techniques* [Hardt, 2011; Dwork and Roth, 2013; Lee and Clifton, 2014], which are a type of differentially private algorithms widely adopted in the literature [Lee and Clifton, 2014; Chen et al., 2015; Li et al., 2015]. They take as input a sequence of queries and a threshold θ, and output either a set of queries whose results are likely to be larger

Algorithm 4.4: BinarySVT (D, $Q = \{q_1, q_2, \ldots\}$, θ, λ)

```
1 compute a noisy version of θ:  θ̂ = θ + Lap(λ);
2 for i = 1, 2, ... do
3 |   compute a noisy version of q_i(D):  q̂_i(D) = q_i(D) + Lap(λ);
4 |   if q̂_i(D) > θ̂ then
5 |   |   output o_i = 1 and continue;
6 |   else
7 |   |   output o_i = 0 and continue;
8 return;
```

than θ [Dwork and Roth, 2013; Lee and Clifton, 2014], or a noisy version of the answers for such a query set [Hardt, 2011]. Intuitively, sparse vector techniques are similar in spirit to PrivTree since they both aim to identify some elements in a set (e.g., a node set or a query set) with "scores" above a given threshold. Motivated by this, in the following, we examine whether sparse vector techniques can be adopted for the hierarchical decomposition problem. Among the three existing variants of sparse vector techniques [Hardt, 2011; Dwork and Roth, 2013; Lee and Clifton, 2014] that satisfy ε-differential privacy[3], we will focus on a variant dubbed *the binary sparse vector technique*, as it is most relevant to our problem. Interested readers are referred to Appendix A for discussions on the other two variants.

Algorithm 4.4 presents a generic version of the binary sparse vector technique. Its input includes (i) a dataset D, (ii) a sequence $Q = \{q_1, q_2, \ldots\}$ of counting queries such that each q_i has sensitivity 1, (iii) a threshold θ, and (iv) a noise scale λ. Its output is a sequence of binary variables $\{o_1, o_2, \ldots\}$, such that $o_i = 1$ indicates that the result of q_i is larger than θ, and $o_i = 0$ indicates otherwise. The algorithm is fairly simple. It first computes a noisy threshold $\hat{\theta} = \theta + \text{Lap}(\lambda)$, and then, for each query q_i in the sequence, it generates a noisy query answer $\hat{q}_i(D)$ using Laplace noise of scale λ (Lines 1-3). If $\hat{q}_i(D) > \theta$, then algorithm outputs $o_i = 1$; otherwise, the algorithm outputs $o_i = 0$ (Lines 4-7). Previous work [Lee and Clifton, 2014] makes the following statement about the privacy assurance of the algorithm:

Statement 4.1 *Algorithm 4.4 ensures ε-differential privacy if $\lambda \geq \frac{2}{\varepsilon}$.*

In other words, the noise scale required by the algorithm is $\Theta(1/\varepsilon)$ and independent of the number of queries.

[3]There exist other variants of sparse vector techniques that satisfy a relaxed version of ε-differential privacy [Hardt and Rothblum, 2010]. We do not consider those variants.

If Statement 4.1 holds, Algorithm 4.4 could yield highly competitive solutions for the problems that we consider. For example, consider the spatial decomposition problem studied in Section 4.2. Given a threshold θ and a set D of spatial points in a multi-dimensional space Ω, we first initialize (i) a quadtree $\mathcal{T}$ containing only a root node v_1 with $\text{dom}(v_1) = \Omega$, and (ii) a query sequence $Q = \{c(v_1)\}$, i.e., Q contains only one query that asks the number of points in $\text{dom}(v_1)$ (we will dynamically append queries to Q during the construction of the quadtree). After the initialization, we invoke the binary sparse vector technique to inspect each query in Q one by one; if the algorithm outputs 1 for a query $c(v)$, then we split the node v in $\mathcal{T}$, and append a query $c(v')$ to the end of Q for each child node v' of v. When all queries in Q are inspected, we return the quadtree $\mathcal{T}$ obtained. By Statement 4.1, $\mathcal{T}$ ensures ε-differential privacy, as long as the binary sparse vector technique uses Laplace noise of scale $\lambda \geq \frac{2}{\varepsilon}$ when generating the noisy versions of $c(v_i)$. In contrast, PrivTree requires injecting Laplace noise of scale $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{1}{\varepsilon} > \frac{2}{\varepsilon}$, which indicates that the solution based on the binary sparse vector technique is more favorable.

Unfortunately, we show that Statement 4.1 does not hold. The error occurs when the proof considers the noisy threshold $\hat{\theta}$ independent of the previous outputs, such that it can be reused on subsequent queries for free. However, this is not true. Each released output actually reveals a portion of information about $\hat{\theta}$, making it less capable in protecting the answer of the next query. In the worst case, Algorithm 4.4 requires $\lambda = \Omega(k/\varepsilon)$ to achieve ε-differential privacy, where k denotes the number of queries.

Lemma 4.4 *There exists a sequence Q of k count queries for which Algorithm 4.4 violates ε-differential privacy if $\lambda \leq \frac{k}{4\varepsilon}$.*

Lemma 4.4 invalidates all solutions based on the binary sparse vector technique, including those in previous work [Lee and Clifton, 2014; Chen et al., 2015; Li et al., 2015]. In concurrent work [Chen and Machanavajjhala, 2015], Chen and Machanavajjhala present a similar analysis on the binary sparse vector technique, and also come to the conclusion that it is not differentially private.

A Corrected Version. Besides the negative results, we also observe that the binary sparse vector technique can be corrected, by limiting the number of outputs to release. Algorithm 4.5 presents the pseudo-code of the revised algorithm, which we refer to as the *corrected sparse vector technique*. Compared with Algorithm 4.4, the corrected version has two differences. First, it takes as input a threshold t and maintains a counter *cnt* of the number of $o_i = 1$ that have been released; whenever the counter reaches t, the algorithm terminates. In other words, at most t instances of $o_i = 1$ will be reported, which ensures that the impact of previous outputs on $\hat{\theta}$ is bounded. Second, in each $q_i(D)$, it injects Laplace noise of scale $t \cdot \lambda$ (instead of λ), which is in accordance with the total number of $o_i = 1$ that might be released. The following lemma shows that Algorithm 4.5 satisfies ε-differential privacy when $\lambda \geq 2/\varepsilon$.

Algorithm 4.5: CorrectedSVT (D, $Q = \{q_1, q_2, \ldots\}$, θ, λ, t)

```
initialize a counter: cnt = 0;
compute a noisy version of θ: θ̂ = θ + Lap(λ);
for i = 1, 2, ... do
    compute a noisy version of q_i(D): q̂_i(D) = q_i(D) + Lap(t · λ);
    if q̂_i(D) > θ̂ then
        output o_i = 1 and continue;
        increase cnt by 1;
        if cnt ≥ t then
            return;
    else
        output o_i = 0 and continue;
return;
```

Lemma 4.5 *Algorithm 4.5 satisfies ε-differential privacy if $\lambda \geq \frac{2}{\varepsilon}$.*

Notice that Algorithm 4.5 requires adding Laplace noise of scale $2t/\varepsilon$ into each query answer. As such, it could not yield a competitive solution for the hierarchical decomposition problem that we consider. For example, suppose that we construct a private quadtree by using the corrected sparse vector technique to choose the nodes that should be split, i.e., the nodes v whose point counts $c(v)$ are larger than a given threshold θ. In that case, we would have to choose an appropriate value for t (i.e., the maximum number of nodes that we can split), which is rather difficult without prior knowledge of the input data. In addition, even if we are able to select an appropriate t, we still need to inject Laplace noise of scale $2t/\varepsilon$ into $c(v)$ when we decide whether v should be split. In contrast, PrivTree only requires Laplace noise of scale $\Theta(1/\varepsilon)$. Therefore, the corrected sparse vector technique is less favorable than PrivTree for hierarchical decomposition.

4.5 Experiments

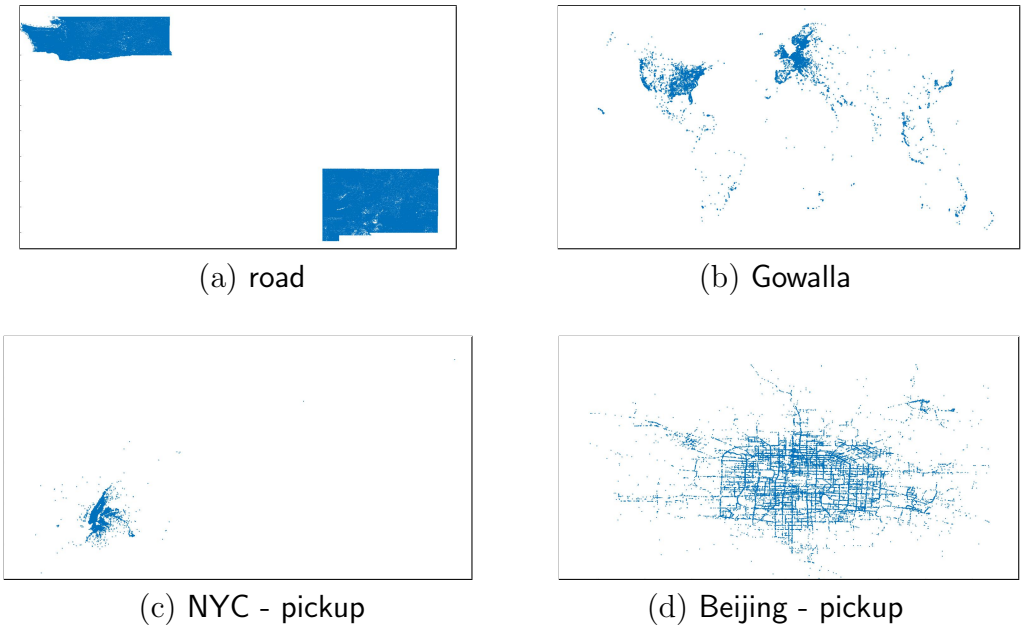
This section evaluates PrivTree against the states of the art on differentially private modelling of spatial and sequence data.

4.5.1 Experiments on Spatial Data

Datasets. We make use of four real spatial datasets shown in Table 4.2: road [Cormode et al., 2012a; Qardaji et al., 2013a], where each point represents the latitude and longitude

Table 4.2: Characteristics of spatial datasets

dataset	dim.	cardinality	description
road	2	1,634,165	coordinates of road intersections in the states of Washington and New Mexico
Gowalla	2	107,091	check-in locations shared by users of a location-based social networking website
NYC	4	98,013	pickup and drop-off locations of NYC taxis
Beijing	4	30,000	pickup and drop-off locations of Beijing taxis



(a) road (b) Gowalla

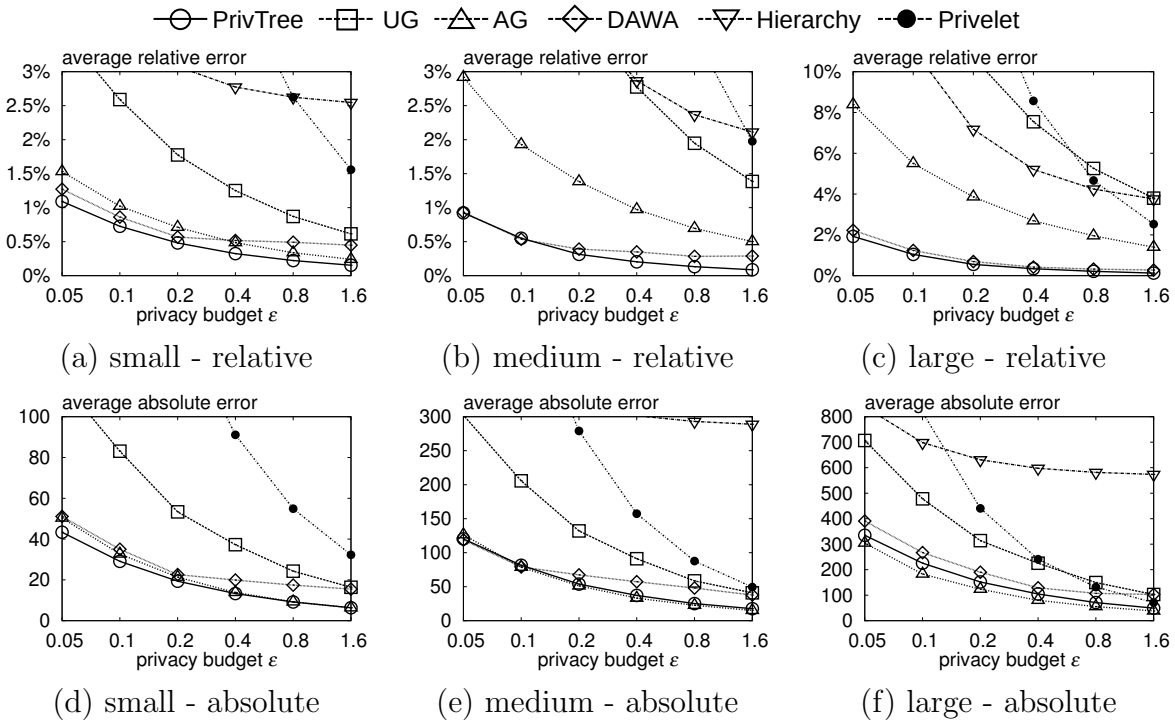
(c) NYC - pickup (d) Beijing - pickup

Figure 4.4: Visualization of datasets

of a road junction in the states of Washington and New Mexico; Gowalla [Qardaji et al., 2013a; Su et al., 2015], which contains check-in locations shared by users on a location-based social networking website; NYC[4] and Beijing[5], which are 4-dimensional datasets that record the pickup and drop-off locations of NYC and Beijing taxis, respectively. Figure 4.4 visualizes the points in road and gowalla, as well as the pickup locations in NYC and Beijing. Observe that the data distribution in road (resp. NYC) is more skewed than that in Gowalla (resp. Beijing).

Methods. We compare PrivTree against five state-of-the-art methods: UG [Qardaji et al., 2013a,b; Su et al., 2015], AG [Qardaji et al., 2013a], Hierarchy [Qardaji et al., 2013b], DAWA [Li et al., 2014a], and Privelet* [Xiao et al., 2011]. UG partitions the data domain into m^d grid cells of equal size, and releases a noisy count for each cell, with

[4]http://publish.illinois.edu/dbwork/open-data/.
[5]http://research.microsoft.com/apps/pubs/?id=152883.



(a) small - relative (b) medium - relative (c) large - relative

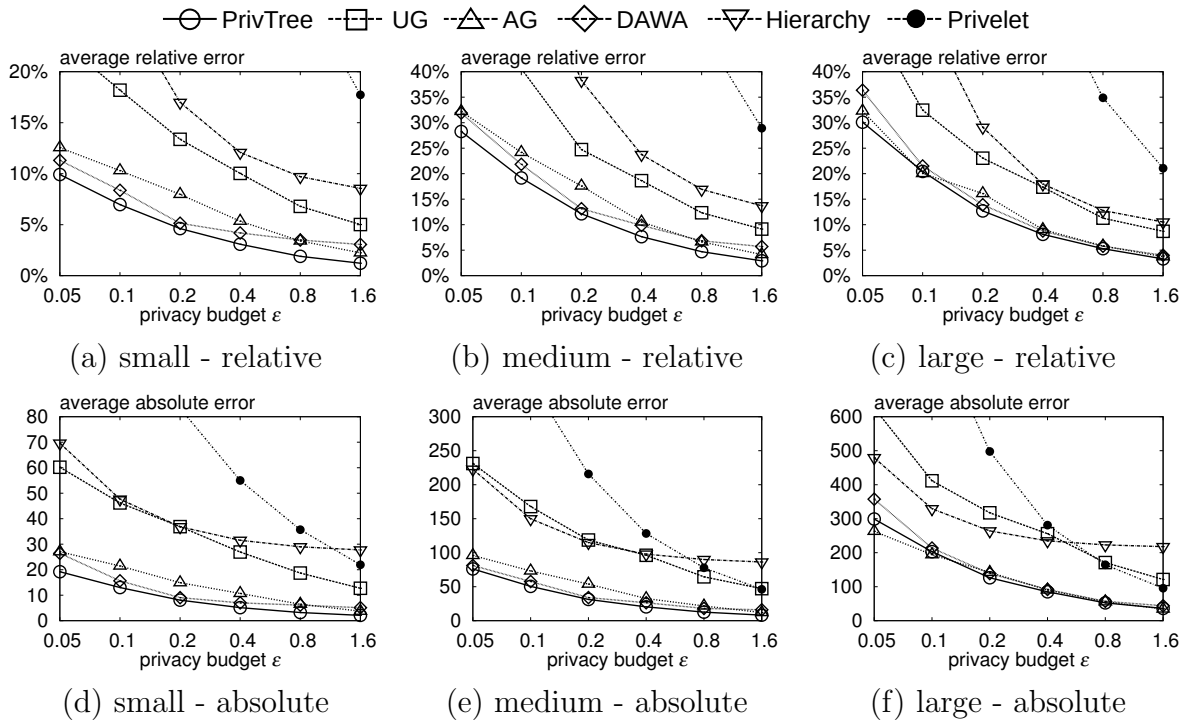
(d) small - absolute (e) medium - absolute (f) large - absolute

Figure 4.5: Range count queries on road

$m = (n\varepsilon/10)^{2/(d+2)}$ [Su et al., 2015]. AG is an improved version of UG that is specifically designed for two-dimensional data. It first employs a coarsened version of UG to produce a set of grid cells; after that, for each cell whose noisy count is above a threshold, AG further splits it into smaller cells and releases their noisy counts. Hierarchy utilizes a multi-level decomposition tree to generate spatial histograms, with the tree height and fanout heuristically chosen to minimize the mean squared error in answering range count queries. DAWA requires as input a workload of range count queries, and it employs the *matrix mechanism* [Li et al., 2010] to generate a histogram that is optimized for the given workload. Privelet* publishes multi-dimensional datasets by utilizing the Haar wavelet transformation to reduce the errors of range count queries.

DAWA and Privelet* both require that the input data should have a discrete domain. Following [Li et al., 2014a], we discretize the domain of each dataset into a uniform grid with 2^{20} cells before feeding it to DAWA and Privelet*. The other parameters of each method (e.g., the height and fanout of the decomposition tree, and the grid granularity) are set as suggested in the original papers. For PrivTree, we set its fanout to 4 (resp. 16) for two-dimensional (resp. four-dimensional) datasets, which is standard for quadtrees.

We adopt the implementations of DAWA and Privelet* provided by their respective authors, and we implement all other methods in C++. All experiments are conducted on a windows/linux machine with a 2.4GHz CPU and 16GB main memory.



(a) small - relative (b) medium - relative (c) large - relative

(d) small - absolute (e) medium - absolute (f) large - absolute

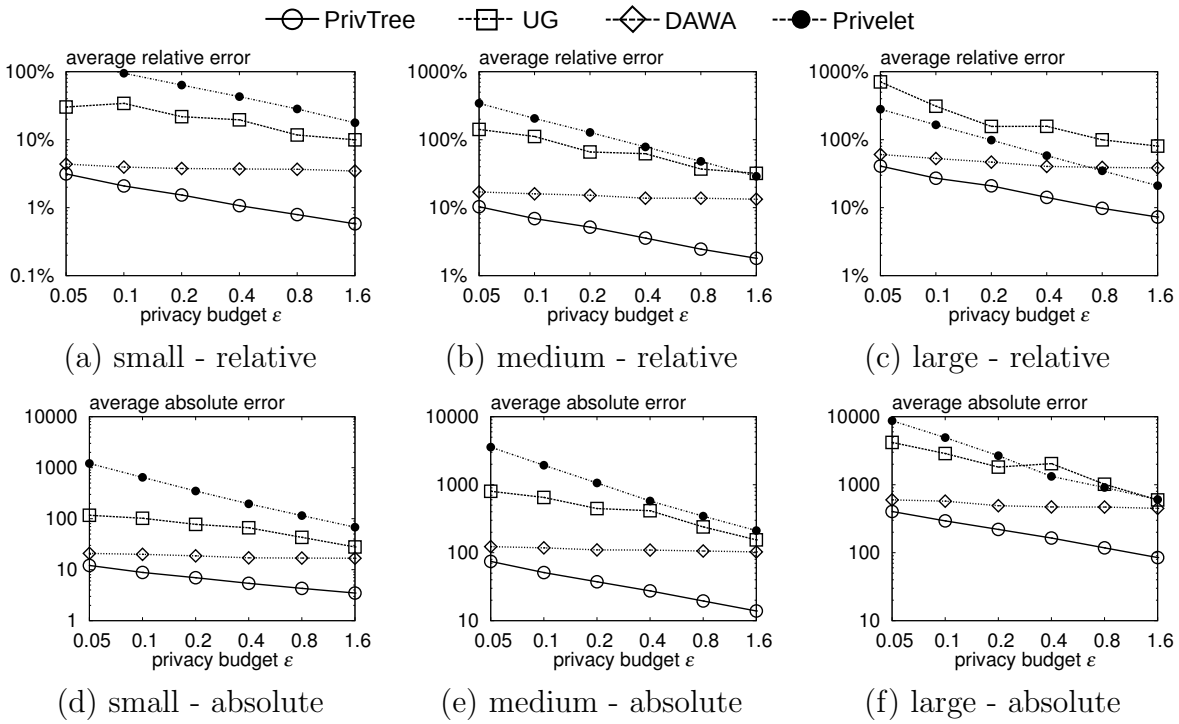
Figure 4.6: Range count queries on Gowalla

Tasks. We apply each method to create private synopses of every dataset, and we evaluate the quality of each decomposition by the accuracy of its answers to range count queries. In particular, we construct three query sets on each dataset: *small*, *medium*, and *large*, each of which contains $10,000$ randomly generated range count queries. Each query in the small, medium, and large set has a region that cover $[0.01\%, 0.1\%)$, $[0.1\%, 1\%)$, and $[1\%, 10\%)$ of the data domain, respectively. Following prior work [Cormode et al., 2012a; Qardaji et al., 2013a], we measure accuracy of an answer $\hat{q}(D)$ to a query q by its *absolute error (AE)* and *relative error (RE)*, defined as

$$AE\left(\hat{q}(D)\right) = \left|\hat{q}(D) - q(D)\right|, \qquad RE\left(\hat{q}(D)\right) = \frac{\left|\hat{q}(D) - q(D)\right|}{\max\left\{q(D), \Delta\right\}},$$

where Δ is a smoothing factor set to 0.1% of the dataset cardinality n [Qardaji et al., 2013a; Xiao et al., 2011]. We repeat each experiment 100 times, and report the average relative error of each method for each query set. For DAWA (which is *query-dependent*), we allow it to generate a synopsis for each query set separately, based on a sample set of 500 queries.

Results. Figures 4.5-4.8 illustrate the average errors of each method on each dataset as a function of the privacy budget ε. The title of each subfigure indicates the query set



(a) small - relative (b) medium - relative (c) large - relative

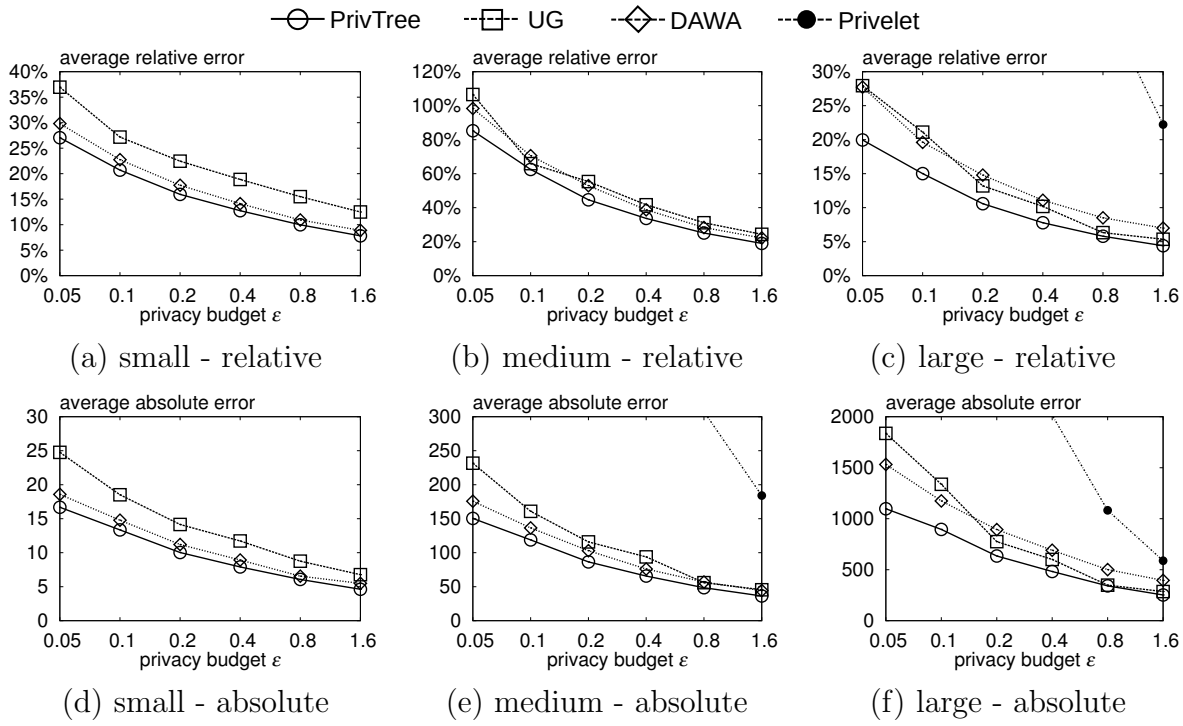
(d) small - absolute (e) medium - absolute (f) large - absolute

Figure 4.7: Range count queries on NYC

and the error measurement in use, e.g., small - relative stands for the small query set and the relative error.

On road, PrivTree significantly outperforms UG, AG, Hierarchy, and Privelet* in terms of relative error, regardless of the query set used and the value of ε. In particular, on the large query set, the average relative error of PrivTree is at most 1/4 (resp. 1/10) of the error of AG (resp. UG and Hierarchy). This demonstrates the effectiveness of PrivTree in approximating the distribution of the input data. To the results in absolute error, we observe some differences. The performance of UG, AG and Privelet* becomes much more charming under the new measurement. On the large query set, AG even outperforms PrivTree by a small margin. To explain, recall that these methods (i.e., UG, Privelet* and the initial step of AG) build their hierarchical structures in a data-independent manner, which leads to (almost) constant error over the query set. In other words, the queries of small counts share the same absolute error to those of large counts. This amplifies the error when the relative measurement is adopted. On the other hand, PrivTree adaptively changes the decomposition tree to fit the data, thus achieving less absolute error for queries over sparse areas. That explains why the superiority of PrivTree is more pronounced in terms of relative error.

Meanwhile, AG is superior to UG and Hierarchy in all cases, which is consistent with the results in previous work [Qardaji et al., 2013a]. Privelet* and Hierarchy are



(a) small - relative (b) medium - relative (c) large - relative

(d) small - absolute (e) medium - absolute (f) large - absolute

Figure 4.8: Range count queries on Beijing

two solutions designed to answer range queries on large domains; therefore, they achieve relatively better utility on the large query set. However, they are the only two methods that fail to adjust their models with the privacy budget ε. Privelet* always decomposes the domain into 2^{20} cells, while Hierarchy uses a decomposition tree of height 3 and fanout 64 (8×8 on two dimensions) in all cases. This is a serious problem: the model of Privelet* is overcomplicated when ε is small, making the true answer overwhelmed by noise; in the meanwhile, the model of Hierarchy is too coarse-grained such that the grid representation of the data loses precision. DAWA is the only method that comes close to PrivTree in both measurements, but its error is never smaller than that of PrivTree, and is 2 to 3 times higher than the latter for the small and medium query sets on road when $\varepsilon \geq 0.8$. Furthermore, we note that DAWA is given a sample query set in advance to optimize its query performance, whereas PrivTree is not given such an advantage.

On Gowalla, PrivTree still consistently achieves the best results, but the performance gaps between PrivTree and the other methods are reduced. The reason is that the data distribution in Gowalla is less skewed than that of road (see Figure 4.4), which makes Gowalla easier to dealt with for all methods. DAWA incurs relatively small errors in all cases, but is noticeably inferior to PrivTree on the small and medium query sets.

On NYC and Beijing, we omit AG and Hierarchy since (i) AG is only applicable on two-dimensional data, and (ii) when applied on a four-dimensional dataset, Hierarchy

Table 4.3: Characteristics of sequence datasets

dataset	$\|\mathcal{I}\|$	cardinality	avg. seq. length	$l^\top$	nr. of seq. with length $> l^\top$
mooc	7	80,362	13.46	50	3,653
msnbc	17	989,818	4.75	20	31,606

produces a decomposition tree with at least 2.18 billion leaf nodes [Qardaji et al., 2013b], which cannot fit in the main memory of our machine. As shown in Figures 4.7-4.8, PrivTree consistently outperforms all other methods by a large margin on the highly skewed NYC, because its tree construction mechanism enables it to effectively adapt to the skewness of the data, by growing the tree tall (resp. short) in the dense (resp. sparse) regions of the data. On the other hand, on the less skewed Beijing, the accuracies of UG and DAWA are considerably improved. Nevertheless, PrivTree still incurs smaller query errors in all settings. One may notice that the error of DAWA on NYC only decreases around 2 times when ε increases from 0.05 to 1.6. We find that it is caused by a "private partitioning" step of DAWA [Li et al., 2014a], as well as the discretization of data domain Ω that it requires. We have tried to reduce this error by using a more fine-grained discretization of Ω, but it leads to an "out of memory" error on our machine.

In summary, PrivTree provides better data utility than all baselines, especially when the input dataset follows a skewed distribution. This makes PrivTree a more favorable approach for releasing spatial data under differential privacy.

4.5.2 Experiments on Sequence Data

Datasets. We use two real sequence datasets: mooc[6] and msnbc [Chen et al., 2012a; Bache and Lichman, 2013]. mooc contains 80,362 learners' behavior sequences on a MOOC platform, and the behaviors are divided into seven categories: working on assignments, watching videos, accessing other course objects, accessing the course wiki, accessing the course forum, navigating to other part of course, and closing the web page. msnbc consists of 989,818 sequences of URL categories, each of which corresponds to a user's browsing history during a 24-hour period on *msnbc.com*. Table 4.3 shows the key statistics of mooc and msnbc. Note that the total number $|\mathcal{I}|$ of symbols in mooc (resp. msnbc) is not excessively large. Otherwise (e.g., when $|\mathcal{I}| > 1000$), the domain of the sequence data would be extremely sparse, in which case it is enormously difficult to publish useful information under differential privacy.

Tasks. We consider two analytical tasks on each sequence dataset D. The first task is to identify the top-k frequent strings in D, i.e., the k strings that appear the largest number of times in the sequences in D. This task is an important primitive in sequence

[6] https://www.kddcup2015.com/

data mining [Han et al., 2011], and is also considered in existing work [Chen et al., 2012a] on sequence data publishing. Following previous work [Chen et al., 2012a], we measure the *precision* of the top-k strings returned by differentially private algorithm, i.e.,

$$\text{precision} = \frac{|K(D) \cap \mathcal{A}(D)|}{k},$$

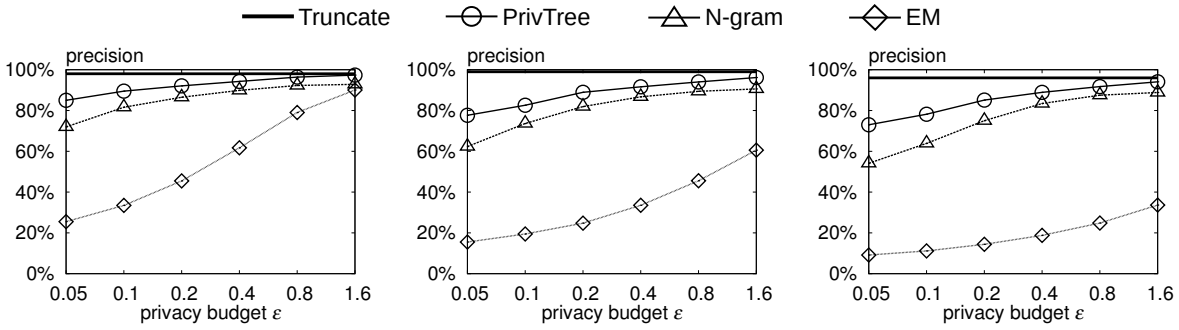
where $K(D)$ is the exact set of top-k frequent strings in D, and $\mathcal{A}(D)$ is the set returned by algorithm $\mathcal{A}$.

The second task is to approximate the distribution of sequence lengths in D. In particular, we apply PrivTree and other existing methods to generate synthetic sequence data from D. Then, we compare the distribution of sequence lengths in the synthetic data with that in D, and we measure their *total variation distance* [Cybakov, 2009a], i.e., half of the L_1 distance between the two probability distributions. For each task, we repeat each experiment 100 times and report the average measurements.

Methods. For the task of top-k frequent string mining, we compare PrivTree against two differentially private techniques: N-gram [Chen et al., 2012a] and the exponential mechanism (EM) [McSherry and Talwar, 2007]. In particular, N-gram is the state-of-the-art solution for sequence data publishing, and it is based on a variable-length n-gram model. N-gram requires a pre-defined threshold n_{max} on the maximum length of n-grams; we set $n_{max} = 5$, as suggested in [Chen et al., 2012a]. Meanwhile, EM is a standard application of the exponential mechanism [McSherry and Talwar, 2007] in our context. It first initializes a set R that contains $|\mathcal{I}|$ string of length 1, each of which consists of a unique symbol in $\mathcal{I}$. After that, it invokes the exponential mechanism k times. In each invocation, it selects the most frequent string r from R with differential privacy, and then replaces r in R with $|\mathcal{I}|$ strings, each of which is obtained by adding a symbol to the end of r. The k strings obtained are then returned as the result. For the task of approximating sequence length distributions, we omit EM since it is inapplicable.

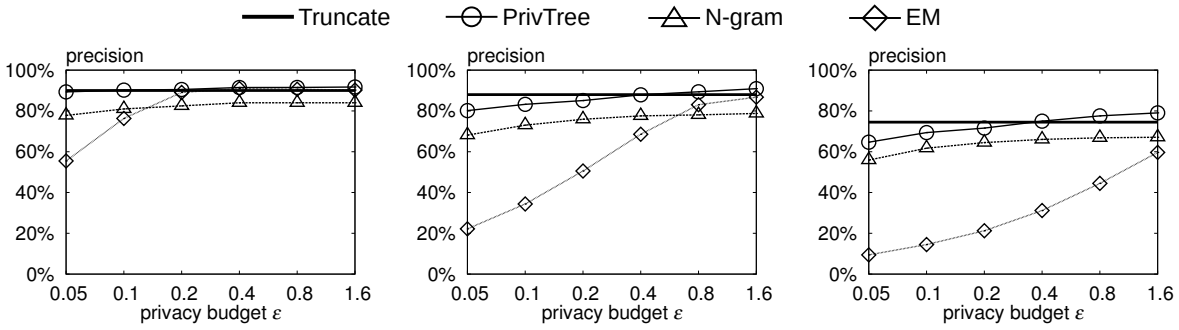
Note that PrivTree, N-gram, and EM all require that the maximum sequence length in the input data is bounded by a constant $l^\top$ that is not excessively large (see Section 4.3.2 for a discussion on the necessity of $l^\top$). Following previous work [Chen et al., 2012a], we set $l^\top$ to be roughly the 95% quantile of the sequence lengths in the input data, i.e., only around 5% sequences are truncated (see Table 4.3). To illustrate the effects of truncation, we also include in our experiments a baseline approach dubbed *Truncate*. This approach directly answers all queries on the truncated dataset, without any privacy assurance.

Results. Figures 4.9-4.10 show the precision of each method (for top-k string mining) as a function of the privacy budget ε. The precision of Truncate remains unchanged for all ε, since it does not enforce differential privacy. Among the differentially private methods,



(a) top50 (b) top100 (c) top200

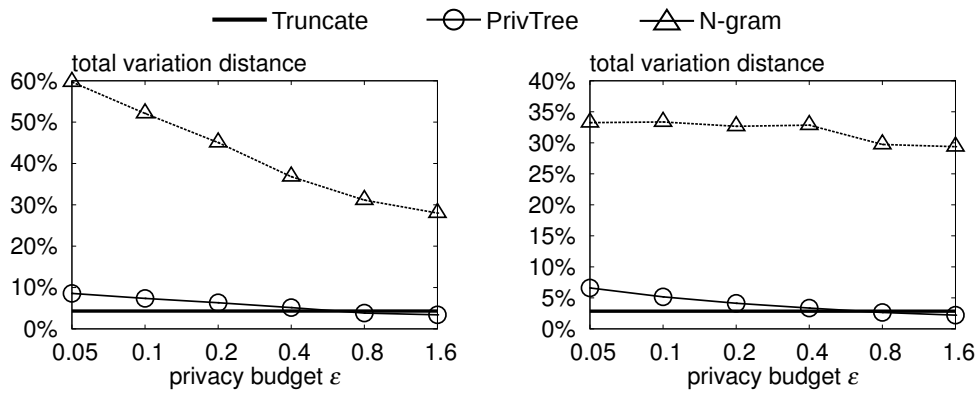
Figure 4.9: Top-k frequent string mining on mooc



(a) top50 (b) top100 (c) top200

Figure 4.10: Top-k frequent string mining on msnbc

PrivTree consistently outperforms N-gram and EM, in most cases by a large margin. Furthermore, in Figure 4.10, PrivTree has an even higher precision than Truncate when $\varepsilon \geq 0.8$. The reason is that the Markov model adopted by PrivTree is able to *recover* some information that is lost due to the truncation of sequences. For example, suppose that the string aa appears in 5 sequences in a dataset D and, in each appearance, it is immediately followed by a symbol b. Assume that one of those 5 sequences (denoted as s) is truncated, and its suffix aab becomes aa after the truncation. In that case, the Markov model can be used able to accurately recover the truncated symbol of s, because, based on the truncated data, it would predict that the "next symbol" after aa is always b. Intuitively, such recovering of information is more effective when the amount of noise in the Markov model is small, which explains why PrivTree outperforms Truncate only when ε is large. In contrast, N-gram never outperforms Truncate, and its precision is lower than that of PrivTree by more than 10% in most settings. In addition, EM yields unattractive precision in almost all cases. Its accuracy degrades with the increases of k, since a larger k requires it to inject more noise into the selection procession of top-k frequent strings.



(a) mooc (b) msnbc

Figure 4.11: Errors of sequence length distributions

In the last set of experiments, we evaluate the accuracy of the sequence length distribution in the synthetic sequence data generated by each method. Figure 4.11 illustrates the total variation distance of each sequence length distribution. Observe that PrivTree incurs a small error comparable to that of Truncate, especially when $\varepsilon \geq 0.2$. In contrast, N-gram entails an enormous error in all cases. Based on the results in Figures 4.9-4.11, we conclude that PrivTree is a more preferable solution than N-gram to modeling sequential data under ε-differential privacy.

4.6 Discussions

One open problem in PrivTree is the selection of $l^\top$ when dealing with sequence data. In the current setting, we assume that $l^\top$ is given as an input. However, the value of $l^\top$ is usually unknown in real applications and should be chosen in a differentially private manner. To the best of our knowledge, a potential solution is to compute a differentially private version of the 95% quantile with the PrivateQuantile Algorithm [Smith, 2011], using a small portion of the privacy budget. However, this algorithm takes as input (i) the number of sequences in the database, and (ii) an upper bound on the length of the sequences, both violating differential privacy if provided directly. Moreover, a natural question is: if an appropriate upper bound on length is given in advance, why not just adopt it as $l^\top$? Therefore, the PrivateQuantile Algorithm is still not the perfect answer to this problem.

Chapter 5

Ladder Framework: Private Release of Graph Data

This chapter presents the ladder framework, our solution to answering subgraph counting queries under differential privacy. Given a subgraph H, e.g., a triangle, we aim to release the number of isomorphic copies of H in the sensitive graph, while protecting individual privacy in the meantime. We write $f_H(g)$ to denote the function that computes the number of copies of subgraph H in graph g. We focus on a number of important classes of H. The function $f_\triangle$ counts the number of triangles in the graph: this is the most heavily studied subgraph counting query, as the triangle is the smallest non-trivial subgraph of interest. Other important subgraph counting functions include: $f_{k\star}$, for counting k-stars (one node of degree k connected to k nodes of degree 1); $f_{k\mathbb{C}}$, for counting k-cliques; and $f_{k\triangle}$ for counting k-triangles (a pair of adjacent nodes i, j that have k common neighbors). Figure 5.1 shows the smallest non-trivial example of each of these classes: note that the 3-clique and the 1-triangle are simply the standard triangle.

We draw up the following list of criteria that our desired solution for differentially private subgraph counting should satisfy:

1. The solution should achieve pure differential privacy;
2. It should be applicable to a wide range of queries, e.g., triangle counting, k-star counting, k-triangle counting and k-clique counting for a constant k;
3. Its time complexity should be no larger than that of computing the true query answer for the queries of interest;
4. It should have high accuracy on its output, bettering that of any previous applicable solution.

Outline of Ladder Framework Development. Our ladder framework achieves all the above criteria. It relies on the careful design of a probability distribution for outputting answers to count queries that tries to maximize the probability of outputting

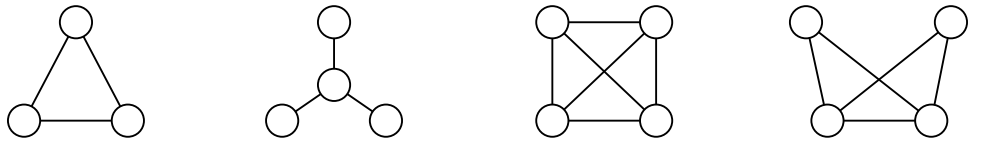


Figure 5.1: Examples of subgraphs

the true answer, or one that is near to it, and minimize the probability of outputting answers that are far from the true answer. The framework of differential privacy places constraints on how these probabilities should behave: the probability of providing output x for input graph g should be "close" (as determined by the parameter ε) to that for input graph g', if g and g' are close (differ by one edge). Rather than go right back to first principles to design these probability distributions, we make use of the exponential mechanism [McSherry and Talwar, 2007], since this approach is general: any differentially private mechanism can be expressed as an exponential mechanism.

Instantiating the exponential mechanism still takes some careful design. We arrive at the design of a symmetric "ladder function" specific to the input graph g, so-called because it is formed from a series of rungs. The first rung groups together a number of outputs that are close to the true answer to $f_H(g)$ (i.e. $f_H(g) \pm 1, f_H(g) \pm 2 \ldots$). The second rung groups together outputs that are a little further away, and so on (see Figure 5.3). The height of each rung determines the probability of outputting the corresponding values, while the width is the number of output values sharing the same height. Ideally, we would make each rung as narrow as possible, to maximize the dropoff in probabilities away from f_H. However, to guarantee differential privacy, we need to make the rungs wide enough so that the ith rung of the ladder for g overlaps with the $i-1$st, ith or $i+1$st rung of the ladder for all neighboring g' (in the language of the exponential mechanism, we seek bounded 'sensitivity' of the derived quality function). We call this the "small leap property".

It turns out that we can design such ladder functions which have rungs that are not too wide but which satisfy the small leap property. Moreover, these can be computed efficiently from the input graph g and used to perform the required output sampling. Our development of these objects takes multiple steps. First, we fill in the necessary details of differential privacy (Section 5.1). Then we show how the use of ladder functions can lead to a practical differentially private solution, if we assume the small leap property, and that the functions converge to a constant rung width (Section 5.2). We still have to show that we can create such functions, and this is the focus of Section 5.2.1.1, where we show that the concept of "local sensitivity at distance t" can be used to create a ladder function with the required properties. This provides a general solution which satisfies privacy; however, some of the quantities required can be time consuming to compute for certain

Table 5.1: Table of notations

notation	description
f, g	the graph query and its sensitive input
$\mathcal{G}_n$	the set of simple graphs with n nodes
$d(g, g')$	minimum edit distance between two graphs
GS	global sensitivity of f
$LS(g)$	local sensitivity of f, given g
$LS(g, t)$	local sensitivity of f at distance t, given g
$q(g, k)$	quality of output k, given input g
Δ_q	global sensitivity of quality function q
$I_t(g)$	the ladder function

subgraph counting queries. Hence, Section 5.3 considers how to instantiate our framework for specific queries, and provides alternate ladder functions for some queries to enable efficient computation. Our experimental evaluation of this approach, and comparison with previous works (where applicable) is detailed in Section 5.4.

5.1 Preliminaries

In this section, we first rewrite the definitions of differential privacy and sensitivity using the concepts and notations developed for graph data, then introduce some basic concepts of local sensitivity. The notational conventions of this chapter are summarized in Table 5.1.

5.1.1 Differential Privacy on Graph

Let $g \in \mathcal{G}_n$ be a sensitive simple graph with n nodes, and f be the subgraph counting query of interest. Differential privacy requires that, prior to $f(g)$'s release, it should be modified using a randomized algorithm $\mathcal{A}$, such that the output of $\mathcal{A}$ does not reveal much information about any edge[1]in g. The definition of differential privacy that we adopt in the context of graphs is as follows:

Definition 5.1 (Neighboring Graphs [Hay et al., 2009]) *Two graphs are neighboring if one of them can be obtained by inserting an edge into the other, i.e., their minimum edit distance [Bunke, 1997]* $d(g, g') \leq 1$.

[1] We adopt edge differential privacy in this chapter, as we are interested in protecting relationship information in a graph. For stronger protection – preserving information associated with nodes – one should seek node differential privacy [Hay et al., 2009]. However, it is open whether any nontrivial graph statistics can be released with node differential privacy.

Definition 5.2 (ε-Differential Privacy [Hay et al., 2009]) *A randomized algorithm $\mathcal{A}$ satisfies ε-differential privacy, if for any two neighboring graphs g and g' and for any possible output O of $\mathcal{A}$,*

$$\Pr[\mathcal{A}(g) = O] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(g') = O]. \tag{5.1}$$

Another important concept in differential privacy is the sensitivity of query f, as defined in Definition 2.4. To distinguish the vanilla sensitivity from its local counterpart (which will be elaborated shortly), we refer to it as the *global sensitivity* in the remainder of the chapter. In our problem setting (i.e., f is a subgraph counting query), the global sensitivity of f is defined as follows:

Definition 5.3 (Global Sensitivity [Dwork et al., 2006b]) *Let f be a function that maps a graph into a real number. The global sensitivity of f is defined as*

$$GS^f = \max_{g,g' | d(g,g') \leq 1} |f(g) - f(g')|, \tag{5.2}$$

where g and g' are any two neighboring graphs.

Last, we review the exponential mechanism [McSherry and Talwar, 2007], as it is a crucial component of the ladder framework. The exponential mechanism releases a differentially private version of $f(g)$, by sampling from f's output domain Ω. The sampling probability for each $k \in \Omega$ is determined based on a user-specified *quality function* q, which takes as input any dataset g and any element $k \in \Omega$, and outputs a numeric score $q(g, k)$ that measures the quality of k: a larger score indicates that k is a better output with respect to g. More specifically, given a graph g, the exponential mechanism samples $k \in \Omega$ from distribution η with probability:

$$\eta(k) \propto \exp\left(\frac{\varepsilon \cdot q(g, k)}{2\Delta_q}\right), \tag{5.3}$$

where Δ_q is defined as the global sensitivity of the quality function, i.e.,

$$\Delta_q = \max_{g,g',k \in \Omega} |q(g, k) - q(g', k)| \text{ for } g, g' \text{ s.t. } d(g, g') \leq 1.$$

The exponential mechanism ensures ε-differential privacy [McSherry and Talwar, 2007].

5.1.2 Local Sensitivity

The scale of noise added by the Laplace mechanism is proportional to GS^f of the query, when the privacy budget ε is fixed. For all queries discussed in this chapter, this approach is not effective since they all have large GS^f relative to the size of the query answer. In [Nissim et al., 2007], the authors seek to address this problem by presenting a local measurement of sensitivity, as in Definition 5.4:

Definition 5.4 (Local Sensitivity [Nissim et al., 2007]) *Let f be a function that maps a graph into a real number. The local sensitivity of f is defined as*

$$LS^f(g) = \max_{g' | d(g,g') \leq 1} |f(g) - f(g')|, \tag{5.4}$$

where g' is any neighboring graph of g.

Note that global sensitivity can be understood as the maximum of local sensitivity over the input domain, i.e., $GS^f = \max_g LS^f(g)$. For simplicity, we write GS and $LS(g)$ instead in the remainder of the chapter, if there is no ambiguity about the subgraph counting query of interest in context.

We refine the definition of local sensitivity, by defining the sensitivity for a particular pair of nodes (i, j), denoted by $LS_{ij}(g)$, to indicate the magnitude of change of f when the connection between i and j is modified (added if absent; deleted if present) in the input graph. This then satisfies $LS(g) = \max_{i,j} LS_{ij}(g)$.

Although tempting, it is not correct to simply substitute $LS(g)$ for GS in the Laplace mechanism. This is because $LS(g)$ itself conveys information from the sensitive data. To address this problem, Nissim et al. [2007] introduce *smooth sensitivity*, an upper bound of $LS(g)$, and show that differential privacy can be achieved by injecting Cauchy noise (instead of Laplace noise) calibrated to smooth sensitivity. As we show in Section 5.4, however, the adoption of Cauchy noise considerably degrades the quality of the output from a differentially private algorithm. This issue could be alleviated by combining smooth sensitivity with Laplace noise, but the resulting solution would only achieve a weakened version of ε-differential privacy (referred to as (ε, δ)-differential privacy [Dwork et al., 2006a]).

In this chapter, we do not use smooth sensitivity, but adopt an intermediate notion of *local sensitivity at distance t* as a crucial part of our solution.

Definition 5.5 (Local Sensitivity at Distance t [Nissim et al., 2007]) *The local sensitivity of f at distance t is the largest local sensitivity attained on graphs at distance at most t from g. The formal definition is:*

$$LS(g, t) = \max_{g^* | d(g,g^*) \leq t} LS(g^*). \tag{5.5}$$

Note that $LS(g, t)$ is a nondecreasing function of t.

By analogy with $LS(g, t)$ and $LS(g)$, we further define $LS_{ij}(g, t)$ as the largest $LS_{ij}(\cdot)$ attained on graphs at distance at most t from g. Then $LS(g, t) = \max_{i,j} LS_{ij}(g, t)$.

Figure 5.2 gives an example of local sensitivity at distance t for triangle counting. The initial graph g in Figure 5.2(a) contains five nodes, three edges and no triangle. Its local sensitivity equals 1, achieved by adding edge (i, j) to the graph, i.e., $LS(g) = LS_{ij}(g) = 1$. Thereafter, $LS(g, 1)$ allows one extra modification before calculating LS. In Figure 5.2(b), we show how the dashed edge (b, j) affects $LS_{ij}(\cdot)$, making $LS(g, 1) = LS_{ij}(g, 1) = 2$. Furthermore, we find that LS could never rise to 3 within 2 steps of modifications, and Figure 5.2(c) illustrates one example achieving $LS(g, 3) = 3$.

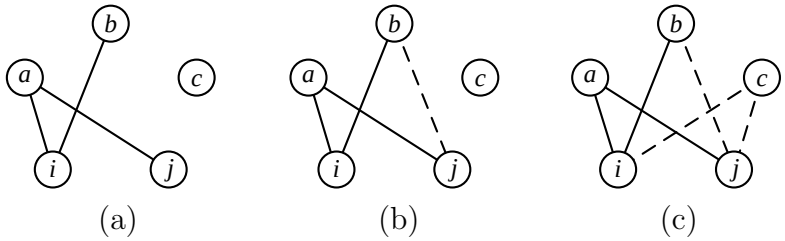


Figure 5.2: Examples of local sensitivity at distance t

5.2 Overview of Solution

This section presents the steps for using our ladder framework for answering subgraph counting queries under ε-differential privacy, given an appropriate function with some assumed properties (these functions are instantiated and proved to possess these properties in subsequent sections). Like the Laplace mechanism, we add a certain amount of noise to the true answer to prevent inference of sensitive information. However, to achieve better results, we aim to exploit the fact that most realistic graphs can tolerate a lower amount of noise while still providing the needed privacy. Towards this end, we need to tailor the noise to the given input, without revealing anything about the input by our choice of noise distribution. We introduce how to build such a distribution through the exponential mechanism in Section 5.2.1, then give an algorithm for efficient sampling in Section 5.2.2.

5.2.1 Building a Solution Distribution

Assume that we are given a query $f_H(g)$ that counts the number of subgraphs H in g. Let η be the distribution of the solution that we return as a private version of $f_H(g)$. In a nutshell, η is a discrete distribution defined over the integer domain, such that $\eta(k)$ increases with $q(g,k)$, where $q(g,k) \leq 0$ is a non-positive integer that quantifies the loss of data utility when we publish k instead of $f_H(g)$. We refer to q as *ladder quality*, since we will use it as a quality function in the exponential mechanism. The expression of $\eta(k)$ as a function of $q(g,k)$ is given in Equation (5.3) of Section 5.1.1.

Our choice of q is determined by a *ladder function* $I_t(g)$, which defines how q varies. Figure 5.3 gives a working example of the q and $I_t(g)$. In particular, q is a symmetric function over the entire integer domain, centered at $f_H(g)$. We define the ladder quality to take its maximum value at $f_H(g)$. Without loss of generality, we set $q(g, f_H(g)) = 0$. For other possible answers, the quality decreases as the distance to the true answer increases. The decreasing speed is controlled by function $I_t(g)$. We refer to each quality level $(0, -1, -2, \ldots)$ as a *rung* of the ladder. In the figure, the first rung, corresponding to a quality level of -1, consists of the $I_0(g)$ integers on each side of $f_H(g)$, whose distances

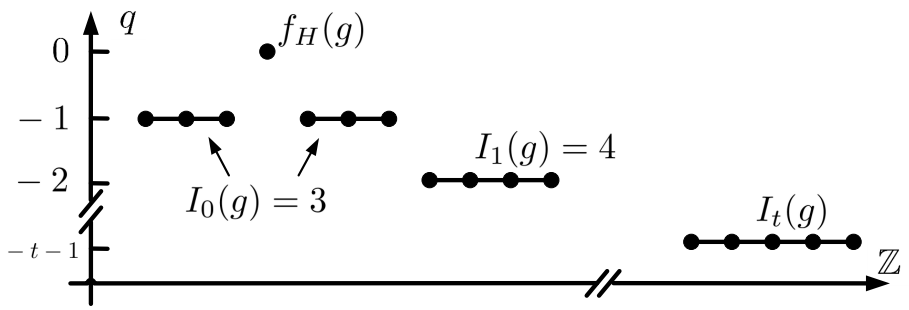


Figure 5.3: An example of a ladder quality

to $f_H(g)$ are no larger than $I_0(g)$. The next rung is formed of the next $I_1(g)$ integers on each side with quality -2, and so on. We say that I_t gives the "width" of the $t+1$st rung. In what follows, we formalize our notions of ladder function and ladder quality.

Definition 5.6 (Ladder Function) *$I_t(g)$ is said to be a ladder function of query f_H if and only if*

(a) $LS^{f_H}(g) \leq I_0(g)$, for any g;

(b) $I_t(g') \leq I_{t+1}(g)$, for any two neighboring graphs g, g', and any nonnegative integer t.

Definition 5.7 (Ladder Quality) *Given a ladder function $I_t(g)$, we define the ladder quality $q(g,k)$ by*

(a) $q(g, f_H(g)) = 0$;

(b) $\forall k \in f_H(g) \pm \left(\sum_{t=0}^{u-1} I_t(g), \sum_{t=0}^{u} I_t(g) \right]$, set $q(g,k) = -u - 1$.

With ladder quality $q(q,k)$, we can assign each integer a quality score. Next we need to determine the global sensitivity of q,

$$\Delta_q = \max_{k,g,g' | d(g,g') \leq 1} |q(g,k) - q(g',k)|,$$

in order to apply the exponential mechanism. For an arbitrary query f_H and a ladder function $I_t(g)$ of f_H, there is a simple answer: $\Delta_q = 1$. By analogy, we find out that if we place two ladders corresponding to neighboring graphs g, g' next to each other and stand on the t'th rung of g's ladder, we can easily "leap across" to g''s ladder, since the rung at the corresponding position is at almost the same height. Formally,

Theorem 5.1 *If $I_t(g)$ is a ladder function then the resulting ladder quality q has $\Delta_q = 1$.*

Proof: To prove this theorem, we must show that for any pair of neighboring graphs g, g', and any integer $k \in \mathbb{Z}$, $|q(g,k) - q(g',k)| \leq 1$ always holds. That is, the quality ascribed to providing k as an output differs by at most 1 for neighboring graphs. In what follows, we discuss three different cases based on how close the value of k is to the top of the ladder.

Special case 1, the center: $k = f_H(g) \Rightarrow q(g,k) = 0$. Given the properties of local sensitivity and ladder function (Definition 5.6(a)) $|k - f_H(g')| \leq LS(g')$ and $LS(g') \leq I_0(g')$, we have

$$f_H(g') - I_0(g') \leq f_H(g) \leq f_H(g') + I_0(g'). \tag{5.6}$$

That is, since k is the query answer for a neighboring graph of g', it must fall within the local sensitivity of the query answer for g'. From Definition 5.7(a) and (b) with $u = 0$ we have $q(g',k) \in \{0, -1\}$. Thus, the difference of quality is bounded by 1.

Special case 2, first rung: If $k \in f_H(g) + (0, I_0(g)]$, then $q(g,k) = -1$: k is on the first rung. By the construction of the ladder function, $q(g',k)$ should be on the center, the first or the second rung, and so the quality changes by at most 1. Formally, the lower bound of k as a function of g' is

$$k > f_H(g) \geq f_H(g') - I_0(g') \quad \text{(from (5.6))};$$

and the upper bound is

$$k \leq f_H(g) + I_0(g) \leq f_H(g') + I_0(g') + I_1(g'),$$

using Definition 5.6(b) to show $I_0(g) \leq I_1(g')$, combined with (5.6). Applying Definition 5.7, the value of the quality function satisfies $q(g',k) \in \{0, -1, -2\}$, and so the bound holds.

General case: Consider $k \in f_H(g) + (\sum_{t=0}^{u-1} I_t(g), \sum_{t=0}^{u} I_t(g)]$ so $q(g,k) = -u-1$, where $u > 0$—we are on the $u+1$st rung on the ladder for g. We argue that we can only climb or descend a single rung when we move to the ladder for g'. The lower bound on k as a function of g' can be obtained using similar steps to case 2 above, from (5.6) and $u-1$ invocations of Definition 5.6(b):

$$\begin{aligned} k > f_H(g) + \sum_{t=0}^{u-1} I_t(g) &= f_H(g) + I_0(g) + \sum_{t=1}^{u-1} I_t(g) \\ &\geq f_H(g') + \sum_{t=0}^{u-2} I_t(g'). \end{aligned}$$

The upper bound uses a similar argument:

$$\begin{aligned} k &\leq f_H(g) + \sum_{t=0}^{u} I_t(g) \\ &\leq f_H(g') + I_0(g') + \sum_{t=1}^{u+1} I_t(g') \;=\; f_H(g') + \sum_{t=0}^{u+1} I_t(g'). \end{aligned}$$

Therefore, $q(g', k)$ is close to $q(g, k) = u - 1$:

$$k \in f_H(g') + \left(\sum_{t=0}^{u-2} I_t(g'), \sum_{t=0}^{u+1} I_t(g') \right] \qquad \Rightarrow q(g', k) \in \{-u, -u-1, -u-2\}.$$

The analysis above proves the result for all $k \geq f_H(g)$. For integers $k < f_H(g)$, the proofs of special case 2 and the general case are symmetric, and yield the required result. □

5.2.1.1 Choices of Ladder Functions

We have explained how to build a solution distribution with the exponential mechanism, given a ladder function $I_t(g)$. In what follows, we consider how best to find such functions in general; we then apply this insight for the functions of interest that count various types of subgraph in the next section.

Some requirements for a good ladder function are as follows. $I_t(g)$ should be a function of input graph g and step t, such that (i) $I_t(g)$ satisfies properties in Definition 5.6, to guarantee the correctness of Theorem 5.1; (ii) $I_t(g)$ is small. The rationale is that $I_t(g)$ controls the shape of the ladder. The smaller $I_t(g)$ the faster the quality decreases away from the true answer, and hence, the exponential mechanism tends to generate a distribution with higher probabilities for outputs closer to the true answer (smaller noise). A straightforward example of a ladder function is the constant function $I_t(g) = GS$, since $LS(g) \leq GS$ for any g, and a constant always satisfies the second requirement. However, as discussed in Section 5.1.2, GS can be extremely large for subgraph counting, which does not adapt to typical graphs, which may not require so much noise. The local sensitivity $I_t(g) = LS(g)$ could be much smaller than GS, but unfortunately is not a ladder function because $LS(g') \leq LS(g)$ does *not* hold for all pairs of neighboring graphs. Our insight for designing good ladder functions is that we can use a relaxation of local sensitivity instead. Specifically, we adopt the notion of local sensitivity at distance t, i.e., $LS(g, t)$. We now prove that this choice is optimal within our framework: local sensitivity at distance t is the *minimum ladder function*, i.e. it is the lower envelope of all possible ladder functions.

Theorem 5.2 *$LS(g, t)$ is the minimum ladder function that satisfies $LS(g, t) \leq I_t(g)$ for any ladder function $I_t(g)$.*

Proof: We first prove that $LS(g,t)$ is a ladder function.

(i) $LS(g) = LS(g,0)$ (meeting Definition 5.6(a));

(ii) $\{g^* \mid d(g',g^*) \leq t\}$ is a subset of $\{g^* \mid d(g,g^*) \leq t+1\}$, given a pair of neighboring graphs g, g' such that $d(g,g') \leq 1$. Therefore, it meets Definition 5.6(b), since:

$$LS(g',t) = \max_{g^* | d(g',g^*) \leq t} LS(g^*) \leq \max_{g^* | d(g,g^*) \leq t+1} LS(g^*) = LS(g,t+1).$$

Next we prove $LS(g,t)$ is no larger than any ladder function $I_t(g)$ by induction on t.

Basis: when $t = 0$, $LS(g,0) = LS(g) \leq I_0(g)$ for all g;

Inductive step: suppose that $LS(g,t) \leq I_t(g)$ holds for all g. It must be shown that $LS(g,t+1) \leq I_{t+1}(g)$ holds for all g. For any pair of neighboring graphs g, g', we have $I_{t+1}(g) \geq I_t(g')$ given $I_t(g)$ is a ladder function. Thus, $I_{t+1}(g) \geq \max_{g' | d(g,g') \leq 1} I_t(g')$ holds for all g. Then given the assumption that $LS(g,t) \leq I_t(g)$ holds for all g,

$$\begin{aligned} I_{t+1}(g) &\geq \max_{g' | d(g,g') \leq 1} I_t(g') \geq \max_{g' | d(g,g') \leq 1} LS(g',t) && \text{by hypothesis} \\ &= \max_{g' | d(g,g') \leq 1} \max_{g^* | d(g',g^*) \leq t} LS(g^*) \\ &= \max_{g^* | d(g,g^*) \leq t+1} LS(g^*) = LS(g,t+1) \end{aligned}$$

thus showing that indeed $LS(g,t+1) \leq I_{t+1}(g)$ holds for all g. By induction, $LS(g,t) \leq I_t(g)$ holds for all g and all nonnegative t. □

$LS(g,t)$ plays an important role in our framework. For certain subgraph counting queries, e.g., $f_\triangle, f_{k\star}$, it is the best ladder function that we can ever find. However, it is not always the preferred choice: as shown in Section 5.3, $LS(g,t)$ can be very hard to compute for some queries. To tackle this problem, we will present how to construct an alternate ladder function (basically, a relaxation of $LS(g,t)$) which is (i) computationally efficient and (ii) still much smaller than GS.

5.2.2 Efficient Sampling from the Distribution

Our solution proceeds by using the ladder quality function to instantiate the exponential mechanism. This results in a solution which is guaranteed to provide differential privacy. However, the output of the mechanism can range over a very large, if not unbounded, number of possibilities, which we must sample from. The naive approach of computing the probability associated with each output value is not feasible: there are too many to calculate. Instead, we need a way to perform sampling in a tractable way. To reduce the time complexity of working with this infinite distribution, we assume a convergence property of $I_t(g)$, that there is a "bottom rung" of our ladder below which all rungs are the same width.

Algorithm 5.1: NoiseSample (f_H, g, ε, I_t): returns f^*

1 compute true answer $f_H(g)$;
2 $range[0] = \{f_H(g)\}$; $weight[0] = 1 \cdot \exp\left(\frac{\varepsilon}{2\Delta_q} \cdot 0\right)$;
3 initialize $dst = 0$;
4 **for** $t = 1$ **to** M **do**
5 $\quad range[t] = f_H(g) \pm (dst, dst + I_{t-1}(g)]$;
6 $\quad weight[t] = 2I_{t-1}(g) \cdot \exp\left(\frac{\varepsilon}{2\Delta_q} \cdot -t\right)$;
7 $\quad dst = dst + I_{t-1}(g)$;
8 $$weight[M+1] = \frac{2C \cdot \exp\left(\frac{\varepsilon}{2\Delta_q} \cdot -(M+1)\right)}{1 - \exp\left(-\frac{\varepsilon}{2\Delta_q}\right)};$$
9 sample an integer $t \in [0, M+1]$ with a probability $weight[t]$ over sum of weights;
10 **if** $t \leq M$ **then**
11 $\quad$ **return** a uniformly distributed random integer in $range[t]$;
12 **else**
13 $\quad$ sample an integer i from the geometric distribution with parameter $p = 1 - \exp\left(-\frac{\varepsilon}{2\Delta_q}\right)$;
14 $\quad$ **return** a uniformly distributed random integer in $f_H(g) \pm \{dst + i \cdot C + (0, C]\}$;

Property 5.1 (Convergence) *$I_t(g)$ converges to a constant C within M steps, i.e., $I_t(g) = C$ for any $t \geq M$.*

With this property, we can more effectively use the exponential mechanism, since there is a bounded number of sampling probabilities. Algorithm 5.1 gives pseudocode for the sampling process under these assumptions, which we now explain. Our algorithm aggregates all integers with the same quality, say $-t$, as t'th rung. The range function (Line 5) describes the domain of each rung, and the weight function (Line 6) gives the sum of weights within the domain. We illustrate how to calculate the range and weight for the first few rungs, e.g., rung 0 (the center) to rung M (Lines 2-7). For other rungs with indices $t > M$, we observe that their weights $2C \cdot \exp(\varepsilon/2\Delta_q \cdot -t)$ form a geometric sequence of t with common ratio $\exp(-\varepsilon/2\Delta_q)$. Therefore, we can write down a closed form sum of weights for all remaining rungs, as in Line 8, considered as one large rung indexed by $M + 1$.

Lines 9-14 perform the random sampling. First, one rung is sampled with a probability proportional to its weight. If it is rung $M + 1$, then we need to further specify the rung index by sampling from a geometric distribution (Line 13). At the end, we return a

random integer within the range of the sampled rung as the final result. The time complexity of Algorithm 5.1 except Line 1 is $O(M)$.

Privacy Guarantee. The correctness of Algorithm 5.1 follows from the correctness of the exponential mechanism [McSherry and Talwar, 2007]. Therefore, we have the following theorem (proof in Appendix B).

Theorem 5.3 *Algorithm 5.1 satisfies ε-differential privacy, provided I_t has Property 5.1.*

5.2.2.1 Convergence of Ladder Functions

This subsection shows how to design a *convergent* ladder function for subgraph counting queries that meets Property 5.1. First, we state a useful lemma for our subsequent analysis. The proof is immediate given the definition of ladder functions, so we omit it.

Lemma 5.1 *If $f(g,t)$ and $h(g,t)$ are ladder functions, $\min\left(f(g,t), h(g,t)\right)$ is a ladder function.*

Using this result, one can easily design a convergent ladder function, e.g., $\min\left(I_t(g), GS\right)$, if the original function $I_t(g)$ is unbounded. This is critical since Algorithm 5.1 requires a convergent function. Unlike $LS(g,t)$ which converges to GS naturally, some ladder functions may need to be explicitly bounded by GS.

Theorem 5.4 *For any subgraph counting query f_H, $\min(I_t(g), GS)$ is a ladder function of f_H that converges to GS within $\binom{n}{2}$ steps, given $I_t(g)$ is a ladder function of f_H.*

Proof: $\min(I_t(g), GS)$ is a ladder function: since $I_t(g)$ and GS are both ladder functions we can invoke Lemma 5.1.

As the distance between any two graphs in $\mathcal{G}_n$ is no larger than $\binom{n}{2}$, we have that $\left\{g^* \mid d(g,g^*) \leq \binom{n}{2}\right\} = \mathcal{G}_n$ holds for any graph $g \in \mathcal{G}_n$. Thus, the local sensitivity at distance $\binom{n}{2}$ equals GS, i.e.,

$$LS\left(g, \binom{n}{2}\right) = \max_{g^* | d(g,g^*) \leq \binom{n}{2}} LS(g^*) = \max_{g^* \in \mathcal{G}_n} LS(g^*) = GS.$$

Given $LS(g,t)$ is the minimum ladder function and is monotonically increasing in t, it follows $I_t(g) \geq LS\left(g,t\right) \geq LS\left(g, \binom{n}{2}\right) = GS$ for any $t \geq \binom{n}{2}$. Hence, $\min(I_t(g), GS) = GS$ for any $t \geq \binom{n}{2}$. □

Recall that the time complexity of Algorithm 5.1 (except Line 1) is linear in M. Therefore we can conclude that the algorithm terminates in $O(n^2)$ time for any subgraph counting query, if a ladder function $I_t(g)$ is given in advance. Quadratic time can still be large, so in the next section, we find ladder functions for classes of subgraphs which converge in $O(n)$ steps.

5.3 Applications

In this section, we show how to apply our framework for a variety of subgraph counting queries, including $f_\triangle, f_{k\star}, f_{k\mathbb{C}}$ and $f_{k\triangle}$. $LS(g,t)$ of the functions $f_\triangle$ and $f_{k\star}$ are carefully studied in [Nissim et al., 2007; Karwa et al., 2014], which can serve as ladder functions for these two queries directly. However, as we will show in this section, computing $LS(g,t)$ can be hard for some queries, e.g., $f_{k\mathbb{C}}$ and $f_{k\triangle}$. Instead of using $LS(g,t)$, we present an efficient method to build a convergent upper bound of $LS(g,t)$, which is shown to satisfy the requirements in Definition 5.6. Such an upper bound could be used as the ladder function for $f_{k\mathbb{C}}$ and $f_{k\triangle}$.

Graph Statistics. Our detailed analysis of global sensitivity, local sensitivity and its upper bound rely on some simple graph statistics related to the (common) neighborhood of nodes.

Definition 5.8 (Graph Statistics) *Let $(x_{ij})_{n\times n}$ be the adjacency matrix of an undirected, simple graph on n nodes. $x_{ij} = 1$ when there exists an edge between nodes i and j, 0 otherwise. Let d_i denote the degree of node i, and $d_m = \max_i d_i$ be the maximum degree of the graph.*

Let A_{ij} be the set of common neighbors of i and j. Node l belongs to A_{ij} if and only if $x_{il}x_{lj} = 1$. Let $a_{ij} = |A_{ij}|$ be the number of common neighbors of i, j, and $a_m = \max_{i,j} a_{ij}$ be the maximum number of common neighbors of a pair of nodes in the graph. Let $b_{ij} = d_i + d_j - 2a_{ij} - 2x_{ij}$ denote the the number of nodes connected to exactly one of the two nodes i, j.

5.3.1 $LS(g,t)$ as Ladder Function

Triangle counting. In Lemma 5.2, we give the global sensitivity and local sensitivity at distance t of triangle counting.

Lemma 5.2 (Claim 3.13 of full version of [Nissim et al., 2007]) *The global sensitivity of $f_\triangle$ is $GS = n - 2$; The local sensitivity at distance t of $f_\triangle$ is $LS(g,t) = \max_{i,j} LS_{ij}(g,t)$, where*

$$LS_{ij}(g,t) = \min\left(a_{ij} + \left\lfloor \frac{t + \min(t, b_{ij})}{2} \right\rfloor, n-2\right).$$

It is easy to prove that $LS(g,t)$ converges to GS when $t \geq 2n$. The time complexity of computing $LS(g,t)$ for $t \in [0, 2n]$ is $O(M(n))$, where $M(n)$ is the time needed to multiply two $n \times n$ matrices.

k-star counting. Another important problem of subgraph counting studied in [Karwa et al., 2014; Chen and Zhou, 2013] is k-star counting. Lemma 5.3 shows its global sensitivity and local sensitivity at distance t.

Lemma 5.3 (Lemma 3.4 of [Karwa et al., 2014]) *The global sensitivity of $f_{k\star}$ is $GS = 2\binom{n-2}{k-1}$; The local sensitivity at distance t of $f_{k\star}$ is $LS(g,t) = \max_{i,j} LS_{ij}(g,t)$, where*

$$LS_{ij}(g,t) = \begin{cases} \binom{\bar{d}_i+t}{k-1} + \binom{\bar{d}_j}{k-1}, & \text{if } t \le B_i; \\ \binom{n-2}{k-1} + \binom{\bar{d}_j+t-B_i}{k-1}, & \text{if } B_i < t \le B_i + B_j; \\ 2\binom{n-2}{k-1}, & \text{if } B_i + B_j < t. \end{cases}$$

Here $\bar{d}_i = d_i - x_{ij}$ and $B_i = n - 2 - \bar{d}_i$. $\bar{d}_j$ and B_j are defined analogously. Without loss of generality, assume that $d_i \ge d_j$.

As with $f_\triangle$, $LS(g,t)$ of $f_{k\star}$ converges to GS when $t \ge 2n$. It takes $O(n \log n + m)$ time to compute $LS(g,t)$ for $t \in [0, 2n]$.

5.3.2 Customized Ladder Function

Although $LS(g,t)$ works well for triangle and k-star counting, it cannot be extended to the class of queries whose $LS(g,t)$ are NP-complete to compute, e.g., k-clique counting for a constant $k > 3$ (proved in Theorem 5.5, the reduction is shown for completeness in Appendix B) and k-triangle counting for $k > 1$ (proved in [Karwa et al., 2014]). To address this problem, the authors of [Karwa et al., 2014] propose to use a *differentially private* version of local sensitivity, to replace the inefficient $LS(g,t)$. However, this method is quite restricted because (i) it is specifically for k-triangle counting; (ii) it achieves (ε, δ)-differential privacy only, and ε is limited to $(0, \frac{3}{2}\ln\frac{3}{2} \approx 0.6]$. In this section, we illustrate how to avoid using $LS(g,t)$ by designing a new customized ladder function.

Theorem 5.5 *Computing $LS(g,t)$ of $f_{k\mathbb{C}}$ for a constant $k > 3$ is NP-complete.*

5.3.2.1 k-clique counting

We first emphasize that we are only interested in a small constant k here, otherwise the counting query $f_{k\mathbb{C}}$ itself is hard to compute. 1- and 2-clique counting are trivial in our setting, and 3-clique (triangle) counting is already well addressed in Lemma 5.2. For other constant $k > 3$, we aim to design a function $I_t(g)$ which satisfies all the requirements in Definition 5.6.

First of all, we introduce some building blocks of $I_t(g)$, i.e., a_m (see Definition 5.8), and the global and local sensitivity of $f_{k\mathbb{C}}$ in the next lemma

Lemma 5.4 *The global sensitivity of $f_{k\mathbb{C}}$ is $GS = \binom{n-2}{k-2}$; the local sensitivity of $f_{k\mathbb{C}}$ is*

$$LS(g) = \max_{i,j} \mathbb{C}(g(A_{ij}), k-2),$$

where $g(S)$ denotes the subgraph induced on g by the node subset S, and $\mathbb{C}(g,k)$ is the number of k-cliques in graph g.

The global sensitivity is achieved when deleting one edge from a complete graph with n nodes. The local sensitivity at (i,j), i.e., $\mathbb{C}(g(A_{ij}), k-2)$, indicates the number of k-cliques that contain nodes i and j when edge (i,j) is added.

Next we give our ladder function for k-clique counting in Theorem 5.6 and explain the intuition of this design in the proof.

Theorem 5.6

$$I_t(g) = \min\left(LS(g) + \binom{a_m + t}{k-2} - \binom{a_m}{k-2}, GS\right)$$

is a ladder function for $f_{k\mathbb{C}}$.

Proof: By Lemma 5.1, we learn that proving $LS(g) + \binom{a_m+t}{k-2} - \binom{a_m}{k-2}$ is a ladder function is sufficient to prove this theorem. The proof contains the following three steps.

(1) $LS(g) \le I_0(g)$, for any g. This step is trivial since $I_0(g) = LS(g)$;

(2) $I_0(g') \le I_1(g)$, for any pair of neighboring graphs g, g'. Let A_{ij} (A'_{ij}) be the set of common neighbors of nodes i,j in g (g'). Let a_m (a'_m) be maximum number of common neighbors in g (g'). Note that the size of A_{ij} is bounded by a_m.

$$\begin{aligned} LS(g') - LS(g) &= \max_{i,j} \mathbb{C}(g'(A'_{ij}), k-2) - \max_{i,j} \mathbb{C}(g(A_{ij}), k-2) \\ &\le \max_{i,j} \left(\mathbb{C}(g'(A'_{ij}), k-2) - \mathbb{C}(g(A_{ij}), k-2)\right). \end{aligned}$$

Recall that there is only one edge difference between g and g'. To increase the number of $(k-2)$-cliques in $g(A_{ij})$ by editing one edge, one can either (i) add one edge within $g(A_{ij})$ while keeping the set A_{ij} unchanged, or (ii) expand A_{ij} by one new node. The maximum increase, i.e., $\binom{a_m}{k-3}$ is achieved in the case that $g(A_{ij})$ is a complete graph of size a_m and $g'(A'_{ij})$ is a complete graph of size $a_m + 1$. However, observe that since

$$\binom{a_m}{k-3} = \binom{a_m+1}{k-2} - \binom{a_m}{k-2},$$

we have

$$LS(g') \le LS(g) + \binom{a_m+1}{k-2} - \binom{a_m}{k-2} \quad \Leftrightarrow \quad I_0(g') \le I_1(g).$$

(3) $I_t(g') \leq I_{t+1}(g)$, for any neighboring graphs g, g' and integer $t > 0$. We make use of the combinatorial identity $\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$ and obtain

$$\begin{aligned} LS(g') + \binom{a'_m + t}{k-2} - \binom{a'_m}{k-2} &= LS(g') + \sum_{i=0}^{t-1} \binom{a'_m + i}{k-3} \\ &\leq LS(g) + \binom{a_m}{k-3} + \sum_{i=0}^{t-1} \binom{a_m + i + 1}{k-3} \\ &= LS(g) + \sum_{i=0}^{t} \binom{a_m + i}{k-3} = LS(g) + \binom{a_m + t + 1}{k-2} - \binom{a_m}{k-2}, \end{aligned}$$

which is equivalent to $I_t(g') \leq I_{t+1}(g)$. The central step of the proof relies on an important property of the maximum number of common neighbors: its global sensitivity equals 1. So it holds that $a'_m - a_m \leq 1$, and so $\binom{a'_m+i}{k-3} \leq \binom{a_m+i+1}{k-3}$. □

This ladder function converges to GS when $t \geq n$. The major computational overhead of $I_t(g)$ for $t \in [0, n]$ is $LS(g)$, which can be computed within $O(n^k)$ time via exhaustive search. In our experiment, we implement a sophisticated algorithm which utilizes the sparsity of the input to improve efficiency. It returns $LS(g)$ within a few seconds for all graphs tested in the next section.

5.3.2.2 k-triangle counting

Besides k-clique counting, we also present k-triangle counting as another example of using customized ladder function. We state the currently known results about k-triangle counting as Lemma 5.5.

Lemma 5.5 (Lemma 4.1 and 4.2 of [Karwa et al., 2014]) *The global sensitivity of $f_{k\triangle}$ is $GS = \binom{n-2}{k} + 2(n-2)\binom{n-3}{k-1}$. The local sensitivity of $f_{k\triangle}$ is*

$$LS(g) = \max_{i,j} \left(\binom{a_{ij}}{k} + \sum_{l \in A_{ij}} \binom{a_{il} - x_{ij}}{k-1} + \binom{a_{lj} - x_{ij}}{k-1} \right);$$

We also have $LS(g') \leq LS(g) + 3\binom{a_m}{k-1} + a_m\binom{a_m}{k-2}$, given a pair of neighboring graphs g and g'.

Similarly, one can design a ladder function for k-triangle counting, as shown in Theorem 5.7. The proof follows the same lines as that of Theorem 5.6.

Theorem 5.7

$$I_t(g) = \min \left(LS(g) + \sum_{i=0}^{t-1} U(a_m + i), GS \right)$$

is a ladder function for $f_{k\triangle}$, where $U(a) = 3\binom{a}{k-1} + a\binom{a}{k-2}$.

Table 5.2: Datasets properties

dataset	nodes	edges	avg. deg.	density	△	3⋆	4ℂ	2△
AstroPh	18,772	198,050	21.101	0.1124%	1.35m	546m	9.58m	72.4m
HepPh	12,008	118,491	19.735	0.1644%	3.36m	1.28b	150m	937m
HepTh	9,877	25,975	5.260	0.0533%	28.3k	2.10m	65.6k	429k
CondMat	23,133	93,439	8.078	0.0349%	173k	37.1m	294k	2.35m
Enron	36,692	183,831	10.020	0.0273%	727k	4.91b	2.34m	36.5m
GrQc	5,242	14,485	5.527	0.1054%	48.3k	2.48m	329k	2.04m

It takes $t \geq 3n$ steps for this ladder function to converge to GS, and the time complexity of computing $I_t(g)$ for $t \in [0, 3n]$ would be $O(n^3)$ using a naive method.

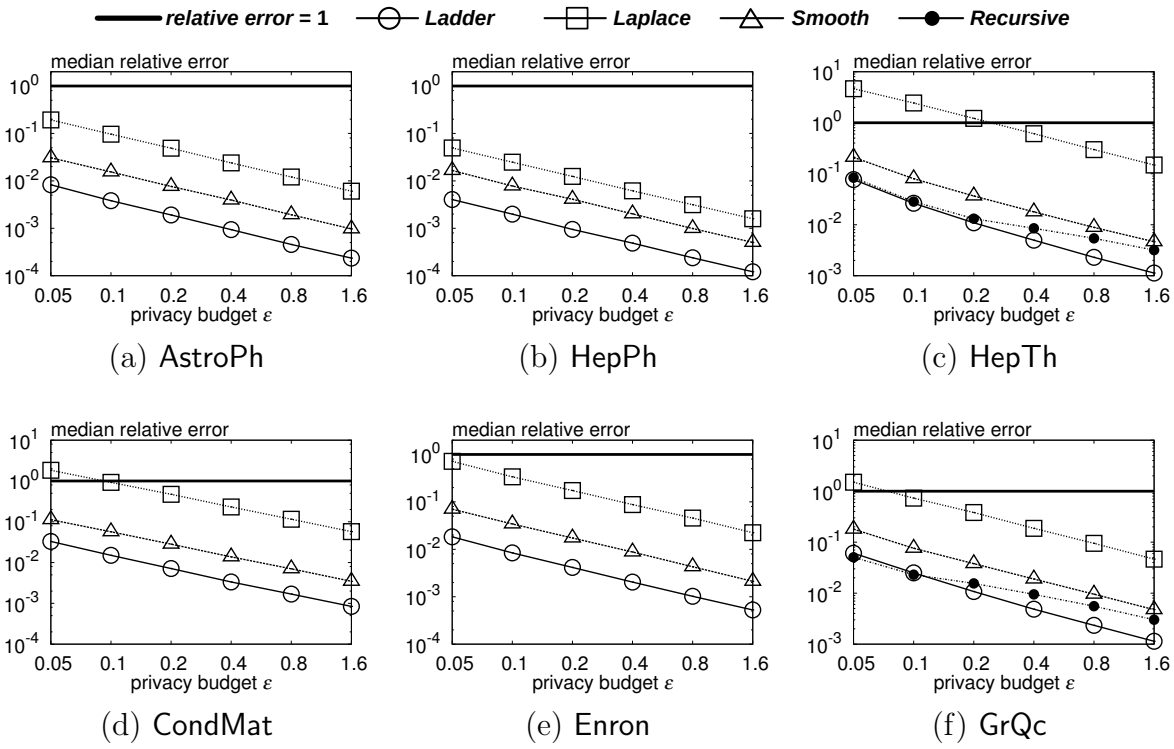
To summarize, we observe some similarities between k-clique and k-triangle counting. First, their $LS(g, t)$ functions are both NP-complete to compute, which motivates our customized ladder functions. Second, there exist efficient ways to compute the local sensitivities $LS(g)$, and $LS(g') - LS(g)$ is bounded by a function of a variable whose global sensitivity is constant, e.g., a_m or d_m. Any subgraph counting query with these two properties could be processed by our framework with a customized ladder function as in Theorems 5.6 and 5.7.

5.4 Experiments

5.4.1 Experimental Settings

Datasets. We make use of six real-world graph datasets in our experiments: AstroPh, HepPh, HepTh, CondMat and GrQc are collaboration networks from the e-print arXiv, which cover scientific collaborations between authors who submitted papers to Astro Physics, High Energy Physics, High Energy Physics Theory, Condensed Matter and General Relativity categories, respectively. In particular, if a paper is co-authored by k authors, the network generates a completed connected subgraph (k-clique) on k nodes representing these authors. Enron is an email network obtained from a dataset of around half a million emails. Nodes of the network are email addresses and if an address i sent at least one email to address j, the graph contains an undirected edge from i to j. Tabel 5.2 illustrates the properties of the datasets and their results to four subgraph counting queries. All datasets can be downloaded from Stanford Large Network Dataset Collection[Leskovec and Krevl, 2014].

Baselines. To justify the performance of our algorithm (denoted as Ladder) in answering subgraph counting queries, we compare it with other four approaches: (i) Laplace [Dwork et al., 2006b], which directly injects Laplace noise with scale GS/ε to the true subgraph



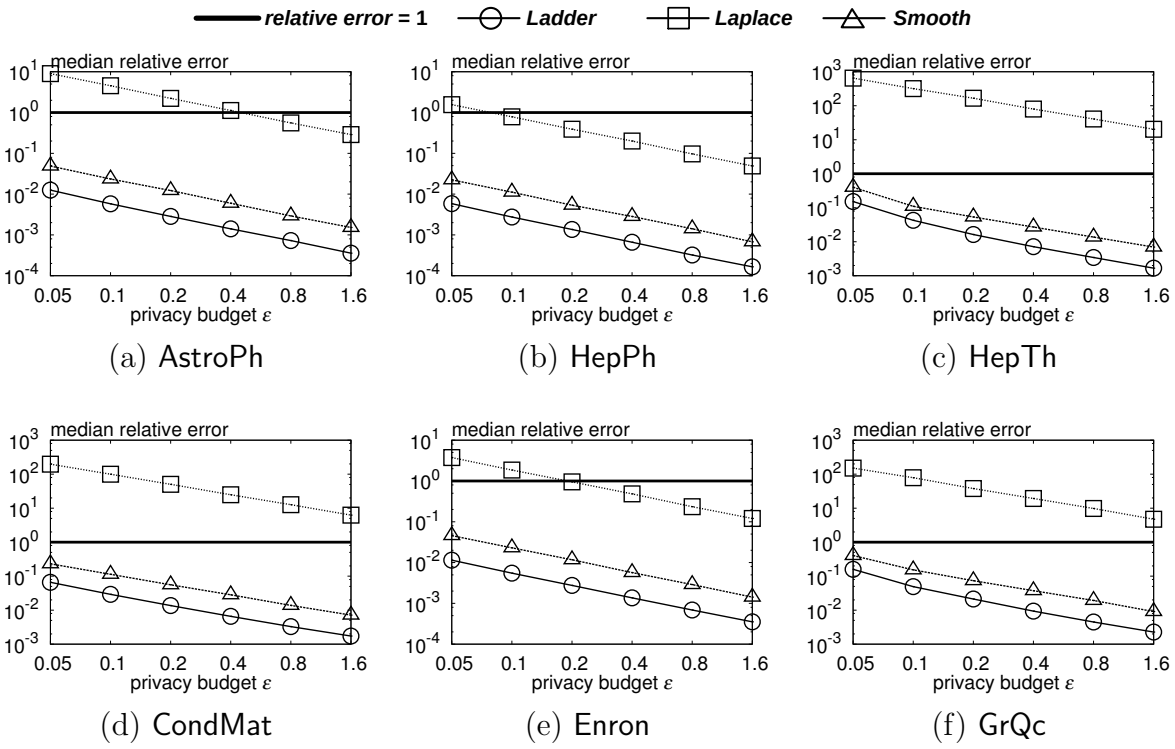
(a) AstroPh (b) HepPh (c) HepTh

(d) CondMat (e) Enron (f) GrQc

Figure 5.4: Triangle counting

counts. (ii) Smooth [Nissim et al., 2007; Karwa et al., 2014], which first computes a smooth upper bound SS of the local sensitivity, then adds Cauchy noise scaled up by a factor of $6SS/\varepsilon$ to the answer. It is only evaluated on two queries $f_{\triangle}$ and $f_{k\star}$ due to the hardness of computing SS for $f_{k\mathbb{C}}$ and $f_{k\triangle}$. (iii) NoisyLS [Karwa et al., 2014], which achieves differential privacy by adding noise proportional to a *differentially private* upper bound on local sensitivity (instead of the smooth upper bound used in Smooth). This approach is designed for $f_{k\triangle}$, and is the only (ε, δ)-differentially private baseline algorithm. The parameter ε is restricted to $(0, \frac{3}{2}\ln\frac{2}{3}]$ while δ is set to a fixed value 0.01 in our experiments. (iv) Recursive [Chen and Zhou, 2013], which answers the query by releasing a differentially private lower bound (which has low-sensitivity) of the counting result. The time complexity of this algorithm is super-quadratic to the number of subgraphs. All experiments using the Recursive mechanism failed to complete within 4 days except for two cases: HepTh-$\triangle$ and GrQc-$\triangle$ since their counting results are relatively small. In contrast, all local sensitivity based solutions (i.e., Ladder, Smooth, NoisyLS) are rather efficient, and in fact share the same time complexity.

Evaluation. We evaluate the performance of Ladder and four baselines on four subgraph counting queries over all six datasets. We measure the accuracy of each method by the *median relative error* [Karwa et al., 2014], i.e., the median of the random variable



(a) AstroPh (b) HepPh (c) HepTh

(d) CondMat (e) Enron (f) GrQc

Figure 5.5: 3-star counting

$|\mathcal{A}(g) - f(g)|/f(g)$ where $\mathcal{A}(g)$ is the differentially private output and $f(g)$ is the true answer. The reason of choosing this measurement is that the mean of Cauchy noise (used in Smooth) is undefined. In other words, the mean error of the Smooth method is never stable, no matter how large the sampling set is. Therefore, in accordance with prior works [Karwa et al., 2014; Chen and Zhou, 2013], we choose to report median error. We also include a bold line to indicate a relative error of 1. Any result above this line is of no practical value. For each result reported, we repeat each experiment 10,000 times to get the median, except for Recursive (where we perform 100 repetitions).

5.4.2 Counting Queries

Figures 5.4-5.7 show all experimental results on counting queries, varying the privacy budget ε from 0.05 to 1.6. The title of each subfigure (e.g., AstroPh) indicates the reported dataset.

Triangle counting. The first query that we evaluate is the triangle counting $f_\triangle$ (see Figure 5.4). In summary, Ladder achieves good accuracy on $f_\triangle$ over all datasets. When privacy budget is relatively large, e.g., $\varepsilon = 1.6$, its median relative error always stays below or close to 0.1%. With the decrease of ε, the accuracy drops but it is still smaller

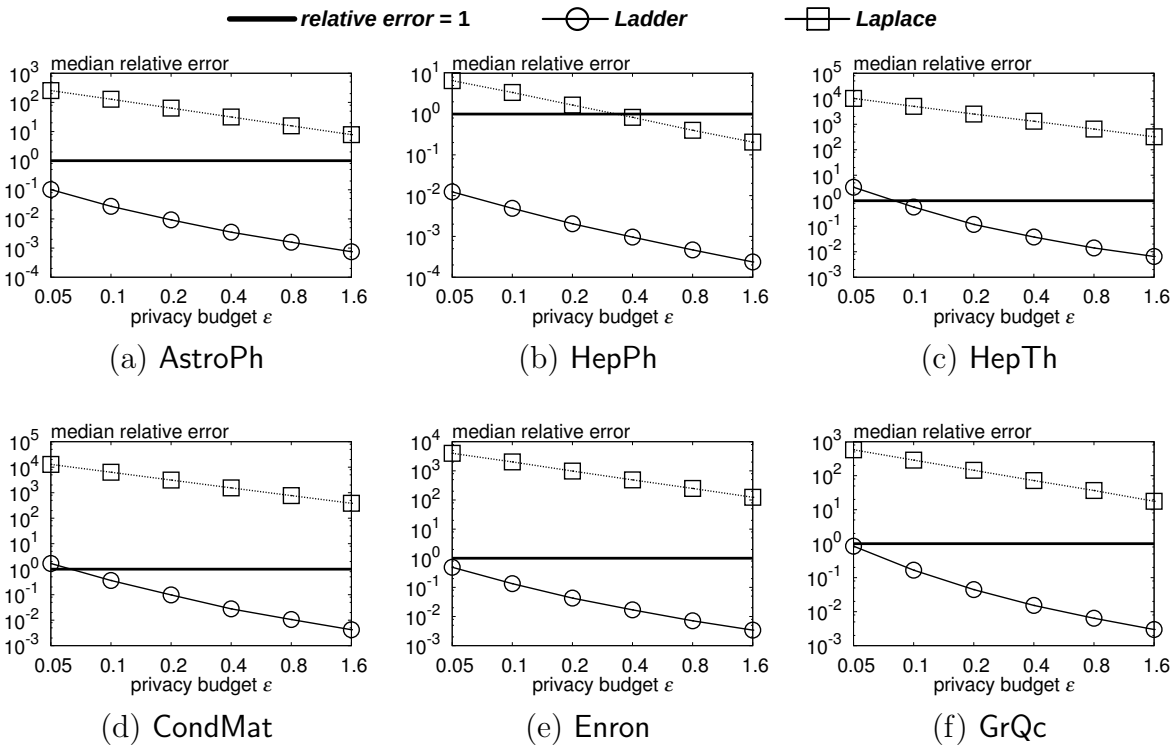
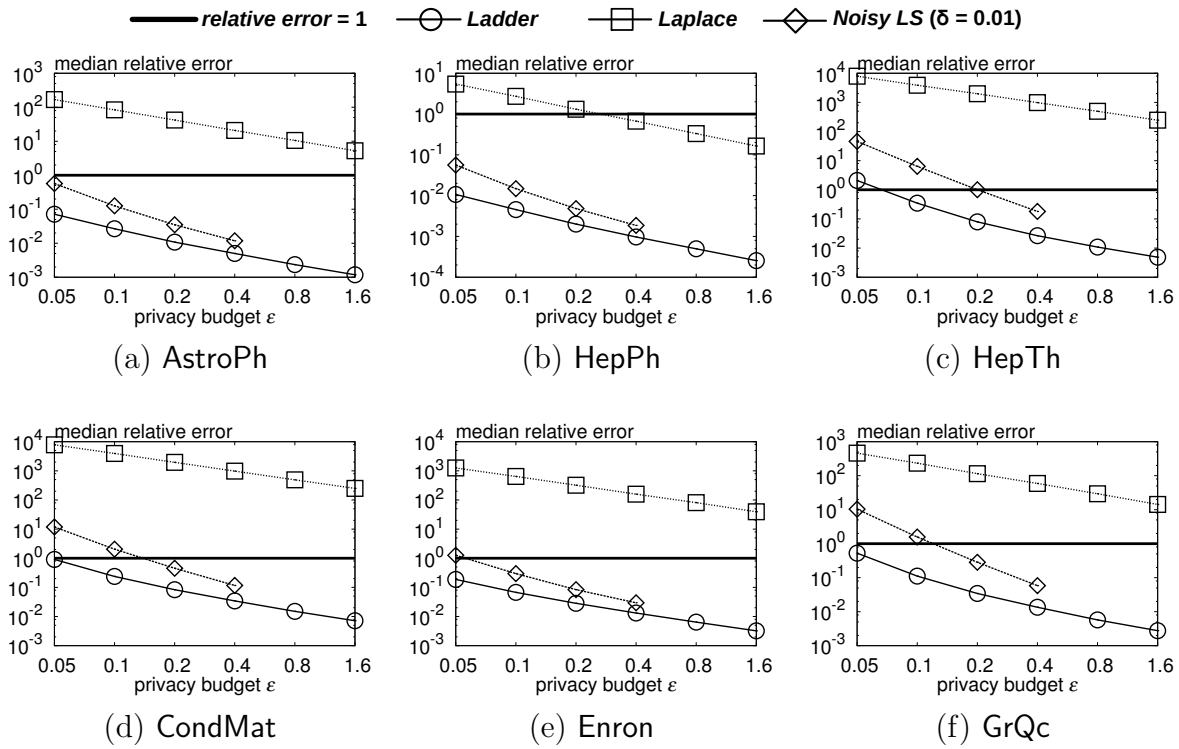


Figure 5.6: 4-clique counting

than 10% even when $\varepsilon = 0.05$. Compared with other differentially private methods, Ladder clearly outperforms Laplace and Smooth in all cases, simply because it injects less noise to the true results. The improvement is significant since the y-axis in shown on a log scale. As for Recursive, it is rather competitive when ε is small, say $\varepsilon \leq 0.1$. However, the gap between Recursive and Ladder begins to grow when ε increases. To explain, recall that the error of Recursive comes from two parts: bias of the lower bound and noise injected to the lower bound. As ε increases, the scale of noise reduces, while the bias is less sensitive to the change of ε then becomes the dominating factor of the error.

k-star counting. Next we evaluate different methods for 3-star counting $f_{3\star}$ in Figure 5.5. Ladder keeps returning extremely accurate results for large ε, and reasonably good results for small ε. Meanwhile, it is still the best solution with an ε-differential privacy guarantee in all settings. In contrast to the case of triangle counting, the performance of Laplace degrades significantly compared to other two local sensitivity based solutions. The main reason is that GS of triangle counting is linear to n while that of 3-star counting is quadratic. On the other hand, the counting results do not increase so dramatically due to the locality of inputs. Therefore the relative error of Laplace increases. Local sensitivity, however, can capture the locality of inputs, leading to a stable relative error when the query is changed.



(a) AstroPh (b) HepPh (c) HepTh

(d) CondMat (e) Enron (f) GrQc

Figure 5.7: 2-triangle counting

k-clique counting. Figure 5.6 presents the results of 4-clique counting ($f_{4\mathbb{C}}$). Ladder is the only private solution besides the naive Laplace, and the former is orders of magnitude better than the latter in terms of accuracy. The lines of Ladder always stay below the bold lines except for two points where ε is extremely small, and the gaps increase markedly with ε. In contrast, the quality of results from Laplace tends to be rather poor, if it is usable at all. Thus we conclude that Ladder is the only effective and efficient algorithm for releasing private k-clique counting.

k-triangle counting. The last query of interest in our experiment is 2-triangle counting $f_{2\triangle}$. We have a new baseline NoisyLS for this query and compare it with Ladder in Figure 5.7. The superiority of Ladder is three-fold: (i) it is always more accurate than NoisyLS when ε varies from 0.05 to 0.4; (ii) it has no extra constraint on privacy budget ε; (iii) it provides stronger privacy protection than NoisyLS. In summary, Ladder is a more preferable solution for private k-triangle counting.

Average degree and density. Besides the comparison among private algorithms, we are also interested in the impact of input graphs to the private counting results. Intuitively, a graph with more edges (assuming a fixed number of nodes) is likely to have more copies of subgraphs and a larger local sensitivity, but the global sensitivity is never a

function of number of edges. Thus, we can expect relatively good performance of Laplace on denser graphs. Nevertheless, the impact of graph density to local sensitivity is still unclear. From Table 5.2, we can learn that AstroPh and HepPh are denser graphs with higher average degree, on which Laplace does have better overall accuracy. Interestingly, local sensitivity based algorithms like Ladder and Smooth also benefit notably from high density inputs, implying that local sensitivity does not increase as drastically as the counting results as the graph gets denser.

5.5 Discussions

The problem of releasing information about private data is a timely one, and there continues to be pressures to make relevant information available in a privacy-respecting fashion. We have shown that the widely-respected differential privacy guarantee can be obtained for graph statistics, specifically subgraph counts, in a way that is efficient and accurate. Future work includes extending the solution to a large scale graph like Facebook. Moreover, the ladder method developed, although tailored for subgraph counts, can be applied more generally, to functions outside the domain of graphs. The main challenge in lifting the notion to other domains is to ensure that concepts such as local sensitivity at distance t are well-defined. It will be natural to study the usefulness of the ladder framework for other structured mathematical objects, such as matrix representations of data, as well as special cases of graphs, such as trees or directed acyclic graphs.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Differential privacy has received a great deal of attention since its proposal. It is well-recognized as the most promising solution for individual privacy protection, due to its mathematical rigor and extremely strong privacy assurance. Among the widely studied problems on differential privacy, we notice that the publication of complex data has not been well addressed by existing techniques. In particular, we investigate the private release of four categories of complex data, namely, multi-dimensional tabular data, spatial data, sequential data and graph data. We show that the differential privacy guarantee can be obtained during the publication of these data, in a way that is efficient and accurate. The approaches are summarized as follows.

In Chapter 3, we propose PrivBayes, a differentially private data publishing algorithm for multi-dimensional datasets. The core idea is to privately build a Bayesian network, which has been proven a powerful way to represent correlated data approximately. We have seen that it is also highly effective as a model to release data while respecting privacy. We see that data released this way is very accurate, and indeed offers better accuracy than customized mechanisms for particular objectives, such as classification. A crucial part of our approach is the crafting of a novel quality function F as a surrogate for mutual information, which dramatically improves the quality of the released data. This requires some efforts in order to compute efficiently, although since this is part of the release process, we can afford to spend more time on this.

In Chapter 4, we study the problem of hierarchical decomposition under differential privacy, and address the central dilemma of choosing the maximum height h of the decomposition tree. We show that the constraint on h can be removed by introducing a carefully controlled bias in deciding when a node should be split. Based on this result, we propose PrivTree, a general approach for hierarchical decomposition on private data, and we showcase its applications on spatial and sequence data release. Our experimental

results demonstrate that PrivTree significantly outperforms the states of the art in terms of data utility.

In Chapter 5, we investigate the problem of releasing sensitive information in graph structured data. We have shown that the widely-respected differential privacy guarantee can be obtained for graph statistics, specifically subgraph counts, in a way that is efficient and accurate. Our experimental study confirms that our method outperforms existing methods for counting triangles and stars in terms of accuracy, and provides solutions for some problems for which no effective method was previously known.

Though each approach above has a different problem setting, they all address the central problem of releasing sensitive data with differential privacy guarantees. Some interactions and similarities can be observed. First, PrivBayes focuses on the correlations among attributes, while PrivTree studies how the domain of each attribute should be split. This makes PrivTree capable as a pre-processing step of PrivBayes. Recall that PrivBayes requires a uniform discretization on each continuous domain, which may lead to significant information loss. In contrast, PrivTree provides a smarter solution to adaptively grouping values in the domain in a data-aware manner. If combined together, they are expected to achieve better utility in private data release. Second, all methods share the same idea of designing alternative functions of small sensitivity, i.e., the score functions F and R in PrivBayes, the score function c (Equation 4.12) for prediction suffix tree in PrivTree, and the ladder quality in ladder framework. These functions are the key components of our algorithms, that help reduce the scale of perturbation in outputs. The intuitions are similar: we utilize the L_1 distance between marginal distributions to avoid large impact on outputs from any individual.

6.2 Future Directions

In this section, we list three potential problems for our future research.

Transactional Data. One future direction is to apply differential privacy on transactional data. In each transactional dataset, there is a universe of items (or symbols) and each data entry, called a transaction, is an unordered collection of items from the universe. For example, a transaction could represent the items purchased by a customer in one visit to a grocery store. Translational data can be used to facilitate a plethora of analysis tasks, among which frequent itemset mining is the most recognized. The discovery of frequent itemsets can serve valuable economic and research purposes [Han et al., 2011], e.g., mining association rules, predicting user behavior, and finding correlations. However, releasing transactional data presents privacy challenges, as the impact of one transaction can be huge on the synthetic output. Although direct application of conventional methods seems to render a differential privacy solution, the quality of

output tends to be rather poor, if usable at all. A possible solution is to follow the idea of partitioning the transactional dataset in a top-down fashion guided by a context-free taxonomy tree, such that the noisy counts of the transactions can be obtained at the leaf nodes [Chen et al., 2011]. This shares quite a few similarities with the application scenario of PrivTree. Therefore, our future work includes to extend PrivTree to find effective solutions to releasing transactional data with differential privacy guarantees.

Text Data. The privacy leak of AOL [Barbaro and Zeller, 2006] is caused by the careless release, in August 2006, of detailed search logs by AOL of a large number of AOL users. To date, most efforts from differential privacy community still fail to manage the private release of text data, e.g., search logs and text comments/recommendations. The challenge of this problem is that text data are always extremely sparse compared to the data domain, leading to the signal being overwhelmed by the noise. Previous techniques [Korolova et al., 2009; Götz et al., 2012] solve only a specific task (e.g., publishing frequent keywords along with counts), and resort to providing a weaker privacy guarantee. Theoretical results in [Götz et al., 2012] even prove that it is impossible to achieve good utility in search log publication, when ε-differential privacy is required. However, we believe that with reasonable modifications to the text data, there exist effective ways to privately release this category of data of tremendous values. One possible direction is to (i) generalize all keywords of input data with a pre-defined taxonomy tree over the keyword dictionary, and (ii) model the correlations among generalized keywords using probabilistic graphical models.

Correlated Data. Another future direction is to extend this work to private datasets that are correlated. One simple but non-trivial example is the publication of a relational database consisting of two tables, and we require the released tables and *their join* to remain as close to the originals as possible. At the first glance, we may address the problem by building a graphical model on the data and then releasing it privately, in a manner similar to PrivBayes. However, care is needed: in the problem setting of PrivBayes we consider, each individual affects at most one single row of the initial database table. As we consider this more complex example, the impact of an individual (and hence the scale of noise needed for privacy) may grow very large with the join operation, and a more careful analysis is needed to ensure that the noise does not outweigh the signal. Fortunately, the current state-of-the-art [Lu et al., 2014] has extended the notion of differential privacy, including the definitions of neighboring datasets and query sensitivity, to the multi-table setting. This work, if combined with PrivBayes, may achieve the accurate release of relational databases with differential privacy guarantees.

Furthermore, the ultimate goal is to release correlated data in a variety of formats. For instance, Facebook is a friendship graph, with demographic characteristics and check-in locations on each node (representing an individual); therefore, the private publication

of the Facebook graph has to take into account the correlations among graph, tabular and spatial datasets. This can be done with PrivBayes, by transforming the spatial and graph properties of each individual into tabular attributes. However, some transformations require significant efforts especially for those relating to graphs. Moreover, the sensitivity of running PrivBayes on the extended table requires careful consideration, as the impact of one individual may spread to multiple rows. In summary, this problem is among the most challenging ones in our future research.

Bibliography

F. Ahmed, R. Jin, and A. X. Liu. A random matrix approach to differential privacy and structure preserved social network graph publishing. *CoRR*, abs/1307.0475, 2013.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.

M. Barbaro and T. Zeller. A face is exposed for AOL searcher no. 4417749. *New York Times*, 2006.

R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, pages 385–421, 2004.

J. Blocki, A. Blum, A. Datta, and O. Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, pages 87–96, 2013.

A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *STOC*, pages 609–618, 2008.

L. Bonomi and L. Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 269–278. ACM, 2013.

H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997.

J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. You might also like: privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, pages 231–246, 2011.

I. P. Castillo. *The Easiest Hard Problem*. 2011.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, page 27, 2011.

K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

R. Chen, N. Mohammed, B. C. Fung, B. C. Desai, and L. Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *CCS*, pages 638–649, 2012a.

R. Chen, B. Fung, B. C. Desai, and N. M. Sossou. Differentially private transit data publication: a case study on the montreal transportation system. In *SIGKDD*, pages 213–221, 2012b.

R. Chen, Q. Xiao, Y. Zhang, and J. Xu. Differentially private high-dimensional data publication via sampling-based inference. In *SIGKDD*, pages 129–138, 2015.

S. Chen and S. Zhou. Recursive mechanism: Towards node differential privacy and unrestricted joins. In *SIGMOD*, pages 653–664, 2013.

Y. Chen and A. Machanavajjhala. On the privacy properties of variants on the sparse vector technique. *CoRR*, abs/1508.07306, 2015.

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

G. Cormode, C. M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012a.

G. Cormode, C. M. Procopiuc, D. Srivastava, and T. T. L. Tran. Differentially private summaries for sparse data. In *ICDT*, pages 299–311, 2012b.

A. B. Cybakov. *Introduction to nonparametric estimation*. Springer, 2009a.

A. B. Cybakov. *Introduction to nonparametric estimation*. Springer, 2009b.

M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000.

B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, pages 217–228, 2011.

I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

C. Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.

C. Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013.

C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503. Springer, 2006a.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006b.

P. Elliot, J. C. Wakefield, N. G. Best, and D. J. Briggs. *Spatial Epidemiology; Methods and applications*. Oxford University Press, 2000.

D. Feldman, A. Fiat, H. Kaplan, and K. Nissim. Private coresets. In *STOC*, pages 361–370, 2009.

A. Friedman and A. Schuster. Data mining with differential privacy. In *SIGKDD*, pages 493–502, 2010.

B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4), 2010.

A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

M. Götz, A. Machanavajjhala, G. Wang, X. Xiao, and J. Gehrke. Publishing search logs - a comparative study of privacy guarantees. *TKDE*, 24(3):520–532, 2012.

F. Gray. Pulse code communication, Mar. 17 1953. URL `https://www.google.com/patents/US2632058`. US Patent 2,632,058.

A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization. In *SODA*, pages 1106–1125, 2010.

J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

M. Hardt. *A study of privacy and fairness in sensitive data analysis*. PhD thesis, Princeton University, 2011.

M. Hardt and A. Roth. Beyond worst-case analysis in private singular vector computation. In *STOC*, pages 331–340, 2013.

M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS*, pages 61–70, 2010.

M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, pages 2348–2356, 2012.

M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. In *ICDM*, pages 169–178, 2009.

M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1):1021–1032, 2010.

A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *EDBT*, pages 123–134, 2010.

V. S. Iyengar. Transforming data to satisfy privacy constraints. In *SIGKDD*, pages 279–288, 2002.

V. Karwa and A. B. Slavković. Differentially private graphical degree sequences and synthetic graphs. In *Privacy in Statistical Databases*, pages 273–285, 2012.

V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *TODS*, 39(3):22, 2014.

S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476. 2013.

D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD*, pages 193–204, 2011.

D. Kifer and A. Machanavajjhala. A rigorous and customizable framework for privacy. In *PODS*, 2012.

D. Kifer, A. D. Smith, and A. Thakurta. Private convex optimization for empirical risk minimization with applications to high-dimensional regression. *Journal of Machine Learning Research - Proceedings Track*, 23:25.1–25.40, 2012.

A. S. Klovdahl, J. J. Potterat, D. E. Woodhouse, J. B. Muth, S. Q. Muth, and W. W. Darrow. Social networks and infectious disease: The colorado springs study. *Social science & medicine*, 38(1):79–88, 1994.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180, 2009.

J. Lee and C. W. Clifton. Top-k frequent itemsets via differentially private fp-trees. In *SIGKDD*, pages 931–940, 2014.

J. Lei. Differentially private m-estimators. In *NIPS*, pages 361–369, 2011.

J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, 2014. URL `http://snap.stanford.edu/data`.

J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012.

C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT*, pages 272–283, 2013.

C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.

C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014a.

H. Li, L. Xiong, and X. Jiang. Differentially private synthesization of multi-dimensional data using copula functions. In *EDBT*, page 475, 2014b.

H. Li, L. Xiong, X. Jiang, and J. Liu. Differentially private histogram publication for dynamic datasets: an adaptive sampling approach. In *CIKM*, pages 1001–1010, 2015.

N. Li, W. Qardaji, D. Su, and J. Cao. PrivBasis: Frequent itemset mining with differential privacy. *PVLDB*, 5(11):1340–1351, 2012.

B.-R. Lin and D. Kifer. Information preservation in statistical privacy and bayesian estimation of unattributed histograms. In *SIGMOD*, pages 677–688, 2013.

W. Lu and G. Miklau. Exponential random graph estimation under differential privacy. In *SIGKDD*, pages 921–930, 2014.

W. Lu, G. Miklau, and V. Gupta. Generating private synthetic databases for untrusted system evaluation. In *ICDE*, pages 652–663, 2014.

K. G. Manton. National long-term care survey: 1982, 1984, 1989, 1994, 1999, and 2004. 2010.

D. Margaritis. *Learning Bayesian Network Model Structure from Data.* PhD thesis, Carnegie Mellon University, 2003.

F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30, 2009.

F. McSherry and R. Mahajan. Differentially-private network trace analysis. In *SIGCOMM*, pages 123–134, 2010.

F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *SIGKDD*, pages 627–636, 2009.

F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.

D. Mir and R. N. Wright. A differentially private estimator for the stochastic kronecker graph model. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 167–176, 2012.

P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: privacy preserving data analysis made easy. In *SIGMOD*, pages 349–360, 2012.

A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, pages 173–187, 2009.

A. Narayanan, H. S. Paskov, N. Z. Gong, J. Bethencourt, E. Stefanov, E. C. R. Shin, and D. Song. On the feasibility of internet-scale author identification. In *IEEE Symposium on Security and Privacy*, pages 300–314, 2012.

K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, pages 75–84, 2007.

D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *PVLDB*, 7(8):637–648, 2014.

W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *ICDE*, pages 757–768, 2013a.

W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, 2013b.

V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, pages 735–746, 2010.

V. Rastogi, M. Hay, G. Miklau, and D. Suciu. Relationship privacy: output perturbation for queries with joins. In *PODS*, pages 107–116, 2009.

D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning*, 25(2-3):117–149, 1996.

B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *Journal of Privacy and Confidentiality*, 4(1):65–100, 2012.

S. Ruggles, K. Genadek, R. Goeken, J. Grover, and M. Sobek. Integrated public use microdata series: Version 6.0, 2015. URL `https://international.ipums.org`.

A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao. Sharing graphs using differentially private graph models. In *IMC*, pages 81–98, 2011.

H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

A. D. Sarwate and K. Chaudhuri. Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data. *IEEE Signal Processing Magazine*, 30(5):86–94, 2013.

A. Smith. Privacy-preserving statistical estimation with optimal convergence rate. In *STOC*, pages 637–648, 2011.

M. Srivatsa and M. Hicks. Deanonymizing mobility traces: using social network as a side-channel. In *CCS*, pages 628–637, 2012.

D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. Differentially private k-means clustering. *CoRR*, abs/1504.05998, 2015.

L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

Y. Wang and X. Wu. Preserving differential privacy in degree-correlation based graph generation. *Trans. Data Privacy*, 6(2):127–145, 2013.

Y. Wang, X. Wu, and L. Wu. Differential privacy preserving spectral graph analysis. In *Advances in Knowledge Discovery and Data Mining*, pages 329–340. 2013.

X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *TKDE*, 23(8):1200–1214, 2011.

Y. Xiao and L. Xiong. Protecting locations with differential privacy under temporal correlations. In *CCS*, pages 1298–1309, 2015.

J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *ICDE*, 2012.

S. Xu, S. Su, X. Cheng, Z. Li, and L. Xiong. Differentially private frequent sequence mining via sampling-based candidate pruning. In *ICDE*, pages 1035–1046, 2015.

G. Yaroslavtsev, G. Cormode, C. M. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. In *ICDE*, pages 745–756, 2013.

G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: Optimizing batch queries under differential privacy. *PVLDB*, 5(11):1352–1363, 2012.

C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *PVLDB*, 6(1):25–36, 2012.

J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *PVLDB*, 5(11):1364–1375, 2012.

J. Zhang, X. Xiao, Y. Yang, Z. Zhang, and M. Winslett. PrivGene: differentially private model fitting using genetic algorithms. In *SIGMOD*, pages 665–676, 2013.

J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via bayesian networks. In *SIGMOD*, pages 1423–1434, 2014.

J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *SIGMOD*, pages 731–745, 2015.

J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*, pages 155–170, 2016.

Appendix A

Additional Analysis on Sparse Vector Techniques

We continue our discussions on the sparse vector techniques, focusing on two existing variants that we referred to as the *vanilla sparse vector technique* [Hardt, 2011] and *reduced sparse vector technique* [Dwork and Roth, 2013], respectively.

Analyzing the Vanilla Sparse Vector Technique. Algorithm A.1 presents a generic version of the vanilla sparse vector technique. The algorithm takes as input five parameters: (i) a dataset D, (ii) a sequence Q of count queries, each of which has sensitivity 1, (iii) a threshold θ, (iv) a noise scale λ, and (v) a positive integer t. The algorithm is largely identical to the binary sparse vector technique (Algorithm 4.4), except for three relatively minor changes. First, instead of outputting a binary variable for each query q_i to indicate whether the noisy query answer $\hat{q}_i(D)$ is above the noisy threshold $\hat{\theta}$, the vanilla sparse vector technique directly outputs $\hat{q}_i(D)$ if it is larger than $\hat{\theta}$, and it outputs a placeholder $\perp$ otherwise. Second, the vanilla sparse vector technique maintains a counter *cnt* of the number of noisy query answers that have been directly output; whenever the counter reaches t, the algorithm terminates. In other words, at most t noisy answers are reported. Third, in each noisy answer $\hat{q}_i(D)$, the vanilla sparse vector technique injects Laplace noise of scale $t \cdot \lambda$ (instead of λ), which is in accordance with the total number of noisy answers that might be released. Previous work [Hardt, 2011] makes the following statement about the privacy guarantee of the vanilla sparse vector technique.

Statement A.1 *Algorithm A.1 satisfies ε-differential privacy if $\lambda \geq \frac{2}{\varepsilon}$.*

In the following, we invalidate Statement A.1 with a counter-example similar to the one used in the proof of Lemma 4.4. Consider three datasets $D_1 = \{a, b\}$, $D_2 = \{a, a, b\}$

Algorithm A.1: VanillaSVT (D, $Q = \{q_1, q_2, \ldots\}$, θ, λ, t)

initialize a counter: $cnt = 0$;
compute a noisy version of θ: $\hat{\theta} = \theta + \mathrm{Lap}(\lambda)$;
for $i = 1, 2, \ldots$ **do**
    compute a noisy version of $q_i(D)$: $\hat{q}_i(D) = q_i(D) + \mathrm{Lap}(t \cdot \lambda)$;
    if $\hat{q}_i(D) > \hat{\theta}$ **then**
        output $o_i = \hat{q}_i(D)$ and **continue**;
        increase cnt by 1;
        if $cnt \geq t$ **then**
            return;
    else
        output $o_i = \perp$ and **continue**;
return;

and $D_3 = \{a, a\}$, among which (D_1, D_2) and (D_2, D_3) are two pairs of neighboring datasets. Let q_a (resp. q_b) be a query that asks for the number of a (reps. b) in a dataset, e.g., $q_a(D_1) = 1$. Let Q consist of k queries, with the first $k-1$ queries being q_a and the last query being q_b.

Suppose that we invoke Algorithm A.1 on D_1, D_2, D_3, respectively, with Q, a noise scale λ, a threshold $\theta = 0$ and $t = 1$. Let E be the event that the Algorithm A.1 outputs the placeholder $\perp$ for the first $k-1$ queries but returns a noisy answer 1 for the last query. That is, $o_i = \perp$ for any $i \in [1, k-1]$, and $o_k = 1$. If Algorithm A.1 satisfies ε-differential privacy, then

$$\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} = \frac{\Pr[D_1 \to E]}{\Pr[D_2 \to E]} \cdot \frac{\Pr[D_2 \to E]}{\Pr[D_3 \to E]} \leq e^{2\varepsilon}.$$

Recall that Algorithm A.1 outputs the noisy answer $\hat{q}(D)$ only if it is larger than the noisy threshold $\hat{\theta}$. Therefore,

$$\begin{aligned}
\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} &= \frac{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\hat{q}_a(D_1) \leq x]\big)^{k-1} \cdot \Pr[\hat{q}_b(D_1) = 1] dx}{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\hat{q}_a(D_3) \leq x]\big)^{k-1} \cdot \Pr[\hat{q}_b(D_3) = 1] dx} \\
&= \frac{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) \leq x - 1]\big)^{k-1} \cdot \Pr[\mathrm{Lap}(\lambda) = 0] dx}{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) \leq x - 2]\big)^{k-1} \cdot \Pr[\mathrm{Lap}(\lambda) = 1] dx}
\end{aligned}$$

Note that in the above equation, we restrict the range of the noisy threshold $\hat{\theta}$ to $(-\infty, 1)$. This is because $\hat{\theta}$ has to be smaller than all noisy answers output by Algorithm A.1, i.e.,

Algorithm A.2: ReducedSVT (D, $Q = \{q_1, q_2, \ldots\}$, θ, λ, t)

```
initialize a counter: cnt = 0;
compute a noisy version of θ: θ̂ = θ + Lap(t · λ);
for i = 1, 2, ... do
    compute a noisy version of q_i(D): q̂_i(D) = q_i(D) + Lap(t · λ);
    if q̂_i(D) > θ̂ then
        output o_i = 1 and continue;
        update the noisy threshold: θ̂ = θ + Lap(t · λ);
        increase cnt by 1;
        if cnt ≥ t then
            return;
    else
        output o_i = 0 and continue;
return;
```

$\hat{\theta} < o_k = 1$. (Previous work [Hardt, 2011] overlooks this issue and considers $\hat{\theta}$ independent of the noisy answers.)

Consider any $\hat{\theta} < 1$. By Equation (2.2),

$$\frac{\Pr[\hat{q}_b(D_1) = 1]}{\Pr[\hat{q}_b(D_3) = 1]} = \frac{\Pr[\mathrm{Lap}(\lambda) = 0]}{\Pr[\mathrm{Lap}(\lambda) = 1]} = e^{1/\lambda}.$$

In addition,

$$\frac{\Pr[\mathrm{Lap}(\lambda) \le x - 1]}{\Pr[\mathrm{Lap}(\lambda) \le x - 2]} = e^{1/\lambda}.$$

It follows that

$$\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} = \frac{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) \le x - 1]\big)^{k-1} \cdot \Pr[\mathrm{Lap}(\lambda) = 0] dx}{\int_{-\infty}^{1} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) \le x - 2]\big)^{k-1} \cdot \Pr[\mathrm{Lap}(\lambda) = 1] dx} = e^{\frac{k}{\lambda}}.$$

Therefore, $\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} > e^{2\varepsilon}$ whenever $\lambda < k/2\varepsilon$. This indicates that, in the worst case, Algorithm A.1 requires Laplace noise of scale $t\lambda = \Omega(\frac{t \cdot k}{\varepsilon})$ (instead of $\frac{2t}{\varepsilon}$) in each noisy answer to ensure ε-differential privacy, where k is the total number of queries and t is the total number of noisy answers returned.

Analyzing the Reduced Sparse Vector Technique. Algorithm A.2 shows the pseudo-code of the reduced sparse vector technique [Dwork and Roth, 2013]. This algorithm is almost identical to the corrected sparse vector technique in Section 4.4, except

that it uses multiple noisy versions of θ with noise scale set to $t \cdot \lambda$ (instead of λ). This change yields less accurate results as it uses a less accurate version of θ to decide whether a query answer is larger than the threshold. Therefore, the algorithm is considered less favorable than the corrected sparse vector technique. Dwork and Roth [2013] prove that Algorithm A.2 satisfies ε-differential privacy when $\lambda \geq 2/\varepsilon$. That is, the scale of Laplace noise in each noisy answer $\hat{q}_i(D)$ should be at least $2t/\varepsilon$.

Appendix B

Proofs

Proof of Lemma 3.1: To calculate the sensitivity of mutual information I, we first model the sensitivity of entropy H. The rationale is that the mutual information between variables X and Π can be rewritten as a function of entropy of X, Π and their joint (X, Π), i.e.,

$$I(X, \Pi) = H(X) + H(\Pi) - H(X, \Pi). \tag{B.1}$$

Therefore, the change of $I(X, \Pi)$ can be decomposed into the changes of $H(X), H(\Pi)$ and $H(X, \Pi)$. Recall the definition of the entropy of variable X:

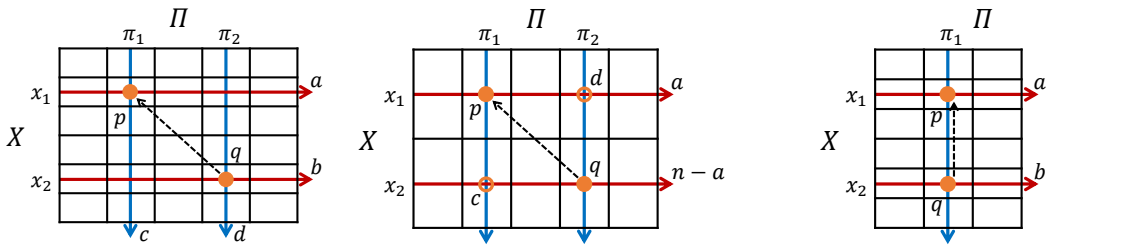
$$H(X) = \sum_{x \in \mathrm{dom}(X)} \Pr[X = x] \log \frac{1}{\Pr[X = x]}.$$

Modifying an arbitrary tuple t in the input will only affect two terms in the above expression. For example, if the value of X in t is modified from x_2 to x_1, the value of $\Pr[X = x_1]$ (resp. $\Pr[X = x_2]$) will increase (resp. decrease) by $1/n$, while others remain unchanged. Formally, let D_1 and D_2 be any pair of neighboring databases, and t_1 and t_2 denote the differing tuples in D_1 and D_2 respectively. Without loss of generality, we assume that $X = x_1$ in tuple t_1, and $X = x_2$ in tuple t_2. Let $H_D(X)$ be the entropy of X given the input database D. We can express the change of $H(X)$ given inputs D_1 and D_2 as follows:

$$\begin{aligned} \Delta H(X) &= H_{D_1}(X) - H_{D_2}(X) \\ &= \left(\frac{a+1}{n} \log \frac{n}{a+1} + \frac{b-1}{n} \log \frac{n}{b-1} \right) - \left(\frac{a}{n} \log \frac{n}{a} + \frac{b}{n} \log \frac{n}{b} \right), \end{aligned} \tag{B.2}$$

where a (resp. b) is the number of tuples with $X = x_1$ (resp. $X = x_2$) in D_2. To further simplify the expression of $\Delta H(X)$, we define a new function

$$f(x) = \frac{x+1}{n} \log \frac{n}{x+1} - \frac{x}{n} \log \frac{n}{x},$$



(a) general case (b) binary case (c) general case (same column)

Figure B.1: Visualization of the change of $I(X, \Pi)$

and revise Equation (B.2) as

$$\Delta H(X) = f(a) - f(b-1). \tag{B.3}$$

In what follows, we list a few properties of the function f:

1. The domain of f consists of integers between 0 and $n-1$ inclusively;

2. $f(x)$ is a monotonically decreasing function over its domain, i.e., $f_{max} = f(0) = \frac{\log n}{n}$ and $f_{min} = f(n-1) = \frac{n-1}{n} \log \frac{n-1}{n}$;

3. $f(x)$ is a convex function, as $f''(x) = \frac{1}{n(x^2+x)} \geq 0$. Therefore, we have lemmas $\max_x [f(x) - f(x+c)] = f(0) - f(c)$ and $\min_x [f(x) + f(c-x)] = 2f(c/2)$, where c is a constant integer in $[0, n-1]$.

General Case. Next we try to express the change of mutual information $I(X, \Pi)$ given neighboring databases D_1 and D_2, as a function of f. Again, let t_1 and t_2 denote the differing tuples in D_1 and D_2 respectively, and we assume that $(X, \Pi) = (x_1, \pi_1)$ in t_1 and $(X, \Pi) = (x_2, \pi_2)$ in t_2. Combined with Equations (B.1) and (B.3), we have

$$\begin{aligned} \Delta I(X, \Pi) &= \Delta H(X) + \Delta H(\Pi) - \Delta H(X, \Pi) \\ &= \Big(f(a) - f(b-1)\Big) + \Big(f(c) - f(d-1)\Big) - \Big(f(p) - f(q-1)\Big), \end{aligned} \tag{B.4}$$

where a (resp. b, c, d, p, q) is the number of tuples with $X = x_1$ (resp. $X = x_2$, $\Pi = \pi_1$, $\Pi = \pi_2$, $(X, \Pi) = (x_1, \pi_1)$, $(X, \Pi) = (x_2, \pi_2)$) in D_2. For a better understanding, Figure B.1(a) visualizes all variables in Equation (B.4).

Computing the sensitivity of $I(X, \Pi)$ is equivalent to finding the maximum value of Equation (B.4). To solve this optimization problem, we need to make use of the properties of function f, and in the mean while, follows all constraints among variables:

Table B.1: An example that achieves the sensitivity of I

$X\backslash\Pi$	0	1	2
0	**1/n**	0	0
1	0	0	$(n-1)/2n$
2	0	$(n-1)/2n$	0

(a) $\Pr[X,\Pi]$ on D_1

$X\backslash\Pi$	0	1	2
0	0	0	0
1	0	0	$(n-1)/2n$
2	0	$(n-1)/2n$	**1/n**

(b) $\Pr[X,\Pi]$ on D_2

- $p \le a \le \min(n-b, n+p-c)$;
- $p \le c \le \min(n-d, n+p-a)$;
- $\max(0, a+c-n) \le p \le \min(a,c)$;
- $q \le b \le \min(n-a, n+q-d)$;
- $q \le d \le \min(n-c, n+q-b)$;
- $\max(1, b+d-n) \le q \le \min(b,d)$.

In what follows, we show how to figure out the maximum of Equation (B.4) in four steps.

Step 1: $a = c = p = 0$**.** Given the property (2) of f, we have $f(a) \le f(0)$ and $f(c) - f(p) \le 0$ as $c \ge p$. Therefore, $\Delta I(X,\Pi) \le f(0) - f(b-1) - f(d-1) + f(q-1)$ and the equation holds when $a = c = p = 0$. In the mean while, by setting a, c, p to 0, the ranges of b, d and q are relaxed to

- $q \le b \le n+q-d$;
- $q \le d \le n+q-b$;
- $\max(1, b+d-n) \le q \le \min(b,d)$.

Step 2: $d = n+q-b$**.** Given the constraint that $d \le n+q-b$ and the property (2) of f, we have $f(d-1) \ge f(n+q-b-1)$. This inequality further extends the upper bound of $\Delta I(X,\Pi)$ to $f(0) - f(b-1) - f(n+q-b-1) + f(q-1)$, which is reachable when $d = n+q-b$. This setting also relaxes the ranges of b and q to $1 \le q \le b \le n$.

Step 3: $q = 1$**.** Given the property (3) of f, we have $f(q-1) - f(q-1+n-b) \le f(0) - f(n-b)$, which makes $\Delta I(X,\Pi) \le 2f(0) - f(b-1) - f(n-b)$ and the equation holds when $q = 1$. The range of b after this step is $1 \le b \le n$.

Step 4: $b = (n+1)/2$**.** The last step utilizes the property (3) of f to minimize $f(b-1) + f(n-b)$. It is easy to know that the minimum value is achieved when $b = (n+1)/2$. Therefore, the final upper bound of Equation (B.4), i.e., the sensitivity of $I(X,\Pi)$, is

$$2f(0) - 2f\left(\frac{n+1}{2}\right) = \frac{2}{n}\log\frac{n+1}{2} + \frac{n-1}{n}\log\frac{n+1}{n-1},$$

which is reachable at $a = 0, b = \frac{n+1}{2}, c = 0, d = \frac{n+1}{2}, p = 0$ and $q = 1$. Table B.1 gives an example that achieves the sensitivity of I. The numbers in bold indicate the differing tuples in D_1 and D_2.

Binary Case. The above discussion analyzes the sensitivity of I without putting any restriction on the domain size of X or Π. A natural question is: does $I(X,\Pi)$ have

Table B.2: An example that achieves the sensitivity of I (binary case)

$X\backslash\Pi$	0	1	2
0	$\mathbf{1/n}$	0	0
1	0	$(n-1)/n$	0

(a) $\Pr[X,\Pi]$ on D_1

$X\backslash\Pi$	0	1	2
0	0	0	0
1	0	$(n-1)/n$	$\mathbf{1/n}$

(b) $\Pr[X,\Pi]$ on D_2

a smaller sensitivity if X or Π is binary? Without loss of generality, we assume that X is binary. Let a (resp. c, d, p, q) be the number of tuples with $X = x_1$ (resp. $(X,\Pi) = (x_2,\pi_1)$, $(X,\Pi) = (x_1,\pi_2)$, $(X,\Pi) = (x_1,\pi_1)$, $(X,\Pi) = (x_2,\pi_2)$) in D_2; see Figure B.1(b) for details. The change of mutual information $\Delta I(X,\Pi)$ in binary case can be expressed as:

$$\begin{aligned}\Delta I(X,\Pi) &= \Delta H(X) + \Delta H(\Pi) - \Delta H(X,\Pi)\\ &= \Big(f(a) - f(n-a-1)\Big) + \Big(f(c+p) - f(d+q-1)\Big) - \Big(f(p) - f(q-1)\Big),\end{aligned}$$

with the following constraints among variables

- $p + d \le a \le n - c - q$;
- $0 \le c \le n - a - q$;
- $0 \le d \le a - p$;
- $0 \le p \le a - d$;
- $1 \le q \le n - a - c$.

Similar to the proof in general case, we solve this optimization problem within four steps of relaxations.

Step 1: $c = p = 0$**.** By the property (2) of f, we have $f(c+p) - f(p) \le 0$ since $c \ge 0$. Therefore, $\Delta I(X,\Pi) \le f(a) - f(n-a-1) - f(d+q-1) + f(q-1)$ and the equation holds when $c = p = 0$. In the mean while, by setting c and p to 0, we relax the ranges of a, d, q to

- $0 \le d \le a \le n - q$;
- $1 \le q \le n - a$.

Step 2: $d = a$**.** Given the constraint that $d \le a$ and the property (2) of f, we have $f(d+q-1) \ge f(a+q-1)$. By setting $d = a$, we extend the upper bound of $\Delta I(X,\Pi)$ to $f(a) - f(n-a-1) - f(a+q-1) + f(q-1)$. This setting also relaxes the ranges of a and q to $0 \le a \le n - q$ and $1 \le q \le n - a$, respectively.

Step 3: $q = 1$**.** Given the property (3) of f, we have $f(q-1) - f(q-1+a) \le f(0) - f(a)$, which makes $\Delta I(X,\Pi) \le f(0) - f(n-a-1)$ and the equation holds when $q = 1$. The range of a after this step is $1 \le a \le n - 1$.

Step 4: $a = 0$**.** In the last step, we simply employ the property (2) of f to maximize $\Delta I(X,\Pi)$, that is $\Delta I(X,\Pi) = f(0) - f(n-1)$ when $a = 0$. Hence, the final upper bound

of $\Delta I(X, \Pi)$ is

$$f(0) - f(n-1) = \frac{1}{n}\log n + \frac{n-1}{n}\log\frac{n}{n-1},$$

which is achieved at $a = c = d = p = 0$ and $q = 1$. Table B.2 presents one such example, in which the numbers in bold indicate the differing tuples in D_1 and D_2. Note that this bound is always no larger than the bound in general case, for any $n \geq 1$.

Other Cases. Last, we discuss other cases that have not been covered yet. First of all, in our analyses of general and binary cases, we assume that the values of X and Π in t_1 and t_2 are different, i.e., $x_1 \neq x_2$ and $\pi_1 \neq \pi_2$. Now we take care of two special cases (i) $x_1 = x_2$ and $\pi_1 = \pi_2$, and (ii) $x_1 \neq x_2$ and $\pi_1 = \pi_2$. We omit the case of $x_1 = x_2$ and $\pi_1 \neq \pi_2$, as it is analogous to (ii). The case (i) is quite straightforward, as it implies no change in the joint distribution $\Pr[X, \Pi]$. Therefore, it is easy to have $\Delta I(X, \Pi) = 0$. The case (ii) requires similar analysis to the general and binary cases. As shown in Figure B.1(c), let a (resp. b, p, q) denote the number of tuples with $X = x_1$ (resp. $X = x_2$, $(X, \Pi) = (x_1, \pi_1)$, $(X, \Pi) = (x_2, \pi_1)$) in D_2. Then, the change of I is

$$\Delta I(X, \Pi) = \Delta H(X) + \Delta H(\Pi) - \Delta H(X, \Pi) = \Big(f(a) - f(b-1)\Big) - \Big(f(p) - f(q-1)\Big),$$

with constraints $0 \leq p \leq a \leq n - b$ and $1 \leq q \leq b \leq n - a$. Following three steps of relaxations, i.e., (i) $a = p = 0$, (ii) $q = 1$ and (iii) $b = n$, we have $\Delta I(X, \Pi) \leq f(0) - f(n-1)$, which is no larger than any upper bound of $\Delta I(X, \Pi)$ in previous cases. Therefore, we complete the proof. □

Proof of Lemma 3.2: The maximum mutual information between variables X and Π is

$$\begin{aligned}\max I(X, \Pi) &= \min\{\max H(X), \max H(\Pi)\} \\ &= \min\{\log|\mathrm{dom}(X)|, \log|\mathrm{dom}(\Pi)|\} = \log|\mathrm{dom}(X)|,\end{aligned}$$

given that $|\mathrm{dom}(X)| \leq |\mathrm{dom}(\Pi)|$. Therefore, the maximum joint distribution for X and Π should be a joint distribution for X and Π with mutual information $\log|\mathrm{dom}(X)|$.

Suppose that $\Pr^{\diamond}(X, \Pi)$ is a joint distribution satisfying the two properties in Lemma 3.2. Given basic results in information theory, the two properties are equivalent to

1. $H(X) = \log|\mathrm{dom}(X)|$;

2. $H(X \mid \Pi) = 0$.

Thus, the mutual information of $\Pr^{\diamond}(X, \Pi)$ is

$$I(X, \Pi) = H(X) - H(X \mid \Pi) = \log |\mathrm{dom}(X)| .$$

By definition, $\Pr^{\diamond}(X, \Pi)$ is a maximum joint distribution.

On the other hand, suppose that $\Pr^{\diamond}(X, \Pi)$ is a maximum joint distribution with mutual information $\log |\mathrm{dom}(X)|$. The mutual information can be expressed as:

$$I(X, \Pi) = \log |\mathrm{dom}(X)| = H(X) - H(X \mid \Pi),$$

where $H(X) \leq \log |\mathrm{dom}(X)|$ and $H(X \mid \Pi) \geq 0$ always hold. Thus, we conclude that I is maximized in the (achievable) case that

1. $H(X) = \log |\mathrm{dom}(X)|$, which is achieved only by the uniform distribution over $\mathrm{dom}(X)$;
2. $H(X \mid \Pi) = 0$, which implies that there is an x for each π such that $\Pr[X = x \mid \Pi = \pi] = 1$.

The above two conditions are equivalent to the properties in the statement of Lemma 3.2. □

Proof of Theorem 3.2: Let $F_D(X, \Pi)$ be the F function for variable X and Π given input database D, i.e,

$$F_D(X, \Pi) = -\frac{1}{2} \min_{\Pr^{\diamond} \in \mathcal{P}^{\diamond}} \|\Pr_D[X, \Pi] - \Pr^{\diamond}[X, \Pi]\|_1.$$

Notice that $\mathcal{P}^{\diamond}[X, \Pi]$ is independent of the input database D. Now consider a pair of neighboring databases D_1 and D_2. We have

$$\|\Pr_{D_1}[X, \Pi] - \Pr_{D_2}[X, \Pi]\|_1 = 2/n. \tag{B.5}$$

Assume that $\Pr \in \mathcal{P}^{\diamond}[X, \Pi]$ is the closest maximum joint distribution to $\Pr_{D_1}[X, \Pi]$. We have

$$F_{D_1}(X, \Pi) = -1/2 \cdot \|\Pr_{D_1}[X, \Pi] - \Pr \|_1.$$

Combined with Equation (B.5), the L_1 distance between $\Pr_{D_2}[X, \Pi]$ and Pr can be upper bounded using the triangle inequality:

$$\begin{aligned}\|\Pr_{D_2}[X, \Pi] - \Pr\|_1 &\leq \|\Pr_{D_1}[X, \Pi] - \Pr\|_1 + \|\Pr_{D_1}[X, \Pi] - \Pr_{D_2}[X, \Pi]\|_1 \\ &= -2 \cdot F_{D_1}(X, \Pi) + 2/n.\end{aligned}$$

On the other hand, recall that Pr is a maximum joint distribution in $\mathcal{P}^{\diamond}[X, \Pi]$. Therefore,

$$\begin{aligned}F_{D_2}(X, \Pi) &= -\frac{1}{2} \min_{\Pr^{\diamond} \in \mathcal{P}^{\diamond}} \|\Pr_{D_2}[X, \Pi] - \Pr^{\diamond}[X, \Pi]\|_1 \\ &\geq -\frac{1}{2} \|\Pr_{D_2}[X, \Pi] - \Pr\|_1 \\ &\geq -\frac{1}{2} \left(-2 \cdot F_{D_1}(X, \Pi) + \frac{2}{n}\right) = F_{D_1}(X, \Pi) - \frac{1}{n}.\end{aligned}$$

Thus, $F_{D_1}(X, \Pi) - F_{D_2}(X, \Pi) \leq 1/n$. □

Proof of Theorem 3.3: We prove this theorem by showing that an instance of the *number partitioning problem* [Castillo, 2011] is polynomial-time reducible to an instance of computing F. The problem asks whether a given multiset S of positive integers can be partitioned into two subsets S_a and S_b, such that the sum of the numbers in S_a equals the sum of the numbers in S_b. The problem is known to be NP-hard [Castillo, 2011]. Given an instance of partition problem, i.e., $S = \{s_1, s_2, \ldots, s_{|S|}\}$, we construct a joint distribution $\Pr[X, \Pi]$ that

(i) $\mathrm{dom}(X) = \{a, b, c, d\}$ and $\mathrm{dom}(\Pi) = \{1, 2, \ldots, |S|\}$;

(ii) Let $m = \sum_{\pi=1}^{|S|} s_\pi$. For each $\pi \in [1, |S|]$, set

$$\Pr[X = a, \Pi = \pi] = \Pr[X = b, \Pi = \pi] = \frac{s_\pi}{2m}$$

and

$$\Pr[X = c, \Pi = \pi] = \Pr[X = d, \Pi = \pi] = 0.$$

Hence, we have

$$\Pr[X = a] = \Pr[X = b] = 0.5$$

and

$$\Pr[X = c] = \Pr[X = d] = 0.$$

This is an eligible input to computing F as all numbers in $\Pr[X, \Pi]$ sum up to 1. By the definition of F in Equation (3.4), computing F is equivalent to finding the minimum L_1 distance between the input and a maximum joint distribution $\Pr^{\diamond}[X, \Pi]$. And $\Pr^{\diamond}[X, \Pi]$ has the following properties:

(i) $\Pr^{\diamond}[X = x] = 1/|\operatorname{dom}(X)| = 0.25$ for any $x \in \operatorname{dom}(X)$;

(ii) For any $\pi \in [1, |S|]$, there is at most one $x \in \operatorname{dom}(X)$ with $\Pr^{\diamond}[X = x, \Pi = \pi] > 0$.

By the property (i) of $\Pr^{\diamond}[X, \Pi]$ and $\Pr[X = c] = 0$, it is easy to calculate the sum of L_1 distance on the cells of $X = c$, that is

$$\sum_{\pi=1}^{|S|} \left| \Pr[X = c, \Pi = \pi] - \Pr^{\diamond}[X = c, \Pi = \pi] \right| = 0.25.$$

The same result applies on the cells of $X = d$.

The sum of L_1 distance on cells of $X = a$ or b requires more careful analysis. Let A (resp. B) be the set of $\pi \in \operatorname{dom}(\Pi)$ such that $\Pr^{\diamond}[X = a, \Pi = \pi] > 0$ (resp. $\Pr^{\diamond}[X = b, \Pi = \pi] > 0$). For convenience, we also abuse notation and define

$$\Pr[A] = \sum_{\pi \in A} \Pr[X = a, \Pi = \pi] \quad \text{and} \quad \Pr[B] = \sum_{\pi \in B} \Pr[X = b, \Pi = \pi].$$

Then the minimum sum of L_1 distance on cells of $X = a$ or b can be expressed as

$$|\Pr[A] - 0.25| + (0.5 - \Pr[A]) + |\Pr[B] - 0.25| + (0.5 - \Pr[B]). \tag{B.6}$$

Given the property (ii) of $\Pr^{\diamond}[X, \Pi]$, we have $A \cap B = \emptyset$. Combined with the following two facts that

(i) $\Pr[X = a, \Pi = \pi] = \Pr[X = b, \Pi = \pi]$ for any $\pi \in \operatorname{dom}(\Pi)$, and

(ii) $\Pr[X = a] = \Pr[X = b] = 0.5$,

we get the bound that $\Pr[A] + \Pr[B] \leq 0.5$. Therefore, the minimum value of Equation (B.6) is achieved if and only if $\Pr[A] = \Pr[B] = 0.25$, which indeed implies an even partition of multiset S. Specifically, for each π in A (resp. B), we add s_π to subset S_a (resp. S_b). The sum of numbers in each subset is $0.25 \cdot 2m = 0.5m$, where m is the sum of all numbers in S. Since that $A \cap B = \emptyset$ and all numbers in S are positive, it is easy to prove that $S_B = S \backslash S_A$. Thus, it is an even partition of S.

Combining all the analyses above with Equation (3.4), we draw the conclusion that if and only if $F(X, \Pi) = -0.5$, there exists an even partition of S. Therefore, computing F is at least as hard as the partition problem. □

Proof of Theorem 3.4: Let $R_D(X, \Pi)$ be the score function for attribute-parent pair (X, Π), when the input database is D, i.e.,

$$R_D(X, \Pi) = \frac{1}{2} \left\| \Pr_D[X, \Pi] - \overline{\Pr}_D[X, \Pi] \right\|_1.$$

Now consider a pair of neighboring databases D_1 and D_2. It is easy to prove that

$$\left\| \Pr_{D_1}[X, \Pi] - \Pr_{D_2}[X, \Pi] \right\|_1 = \frac{2}{n}.$$

Combined with this equation and the triangle inequality, the sensitivity of R can be rewritten as follows:

$$\begin{aligned}
&2 \cdot \left(R_{D_1}(X, \Pi) - R_{D_2}(X, \Pi)\right) \\
&= \|\Pr_{D_1}[X, \Pi] - \overline{\Pr}_{D_1}[X, \Pi]\|_1 - \|\Pr_{D_2}[X, \Pi] - \overline{\Pr}_{D_2}[X, \Pi]\|_1 \\
&\leq \|\Pr_{D_1}[X, \Pi] - \Pr_{D_2}[X, \Pi]\|_1 + \|\overline{\Pr}_{D_1}[X, \Pi] - \overline{\Pr}_{D_2}[X, \Pi]\|_1 \\
&= \frac{2}{n} + \|\overline{\Pr}_{D_1}[X, \Pi] - \overline{\Pr}_{D_2}[X, \Pi]\|_1. \qquad \text{(B.7)}
\end{aligned}$$

Next we discuss how to bound the second half of Equation (B.7). Without loss of generality, let t_1 and t_2 denote the only differing tuples in D_1 and D_2 respectively, and assume that $(X, \Pi) = (x_1, \pi_1)$ on t_1 and $(X, \Pi) = (x_2, \pi_2)$ on t_2. Then we have

- $\Pr_{D_1}[X = x_1] = \Pr_{D_2}[X = x_1] + \frac{1}{n}$;
- $\Pr_{D_1}[X = x_2] = \Pr_{D_2}[X = x_2] - \frac{1}{n}$;
- $\Pr_{D_1}[\Pi = \pi_1] = \Pr_{D_2}[\Pi = \pi_1] + \frac{1}{n}$;
- $\Pr_{D_1}[\Pi = \pi_2] = \Pr_{D_2}[\Pi = \pi_2] - \frac{1}{n}$.

Therefore, only the probabilities at positions (x, π) that $x \in \{x_1, x_2\}$ or $\pi \in \{\pi_1, \pi_2\}$ are different between $\overline{\Pr}_{D_1}[X, \Pi]$ and $\overline{\Pr}_{D_2}[X, \Pi]$. We can further divide these positions into two groups: (i) the ones with $x \in \{x_1, x_2\}$ and $\pi \in \{\pi_1, \pi_2\}$, and (ii) the rest.

To the positions in group (ii), computing the sum of difference is quite straightforward. Take positions with $x = x_1$ as an example. We have

$$\begin{aligned}
&\sum_{\pi \notin \{\pi_1, \pi_2\}} \left| \overline{\Pr}_{D_1}[X = x_1, \Pi = \pi] - \overline{\Pr}_{D_2}[X = x_1, \Pi = \pi] \right| \\
&= \sum_{\pi \notin \{\pi_1, \pi_2\}} \left| \Pr_{D_1}[X = x_1] \Pr_{D_1}[\Pi = \pi] - \Pr_{D_2}[X = x_1] \Pr_{D_2}[\Pi = \pi] \right| \\
&= \frac{1}{n} \cdot \sum_{\pi \notin \{\pi_1, \pi_2\}} \Pr_{D_1}[\Pi = \pi].
\end{aligned}$$

The same result can be obtained for positions with $x = x_2$. In the case of $\pi = \pi_1$ or π_2, the sum of difference is bounded by $\frac{1}{n} \cdot \sum_{x \notin \{x_1, x_2\}} \Pr_{D_1}[X = x]$. On the other hand, there are only four positions in group (i), and again we take one of them, i.e., $(X, \Pi) = (x_1, \pi_1)$ as an example:

$$\begin{aligned}
&\left|\overline{\Pr}_{D_1}[X = x_1, \Pi = \pi_1] - \overline{\Pr}_{D_2}[X = x_1, \Pi = \pi_1]\right| \\
&= \left|\Pr_{D_1}[X = x_1]\Pr_{D_1}[\Pi = \pi_1] - \Pr_{D_2}[X = x_1]\Pr_{D_2}[\Pi = \pi_1]\right| \\
&= \left|\Pr_{D_1}[X = x_1]\Pr_{D_1}[\Pi = \pi_1] - (\Pr_{D_1}[X = x_1] - \frac{1}{n})(\Pr_{D_1}[\Pi = \pi_1] - \frac{1}{n})\right| \\
&\leq \frac{1}{n} \cdot \Pr_{D_1}[X = x_1] + \frac{1}{n} \cdot \Pr_{D_1}[\Pi = \pi_1] + \frac{1}{n^2}
\end{aligned}$$

Putting all bounds together, we get an upper bound for $\|\overline{\Pr}_{D_1}[X, \Pi] - \overline{\Pr}_{D_2}[X, \Pi]\|_1$ as follows.

$$\begin{aligned}
\left\|\overline{\Pr}_{D_1}[X, \Pi] - \overline{\Pr}_{D_2}[X, \Pi]\right\|_1 &\leq \frac{2}{n} \cdot \sum_{\pi} \Pr_{D_1}[\Pi = \pi] + \frac{2}{n} \cdot \sum_{x} \Pr_{D_1}[X = x] + \frac{4}{n^2} \\
&= \frac{4}{n} + \frac{4}{n^2}.
\end{aligned}$$

Thus, $R_{D_1}(X, \Pi) - R_{D_2}(X, \Pi) \leq \frac{3}{n} + \frac{2}{n^2}$. □

Proof of Lemma 4.1: By Equations (2.2) and (4.4), for any x,

$$\rho(x) = \ln\left(\frac{\int_{\theta-x}^{+\infty} \frac{1}{2\lambda} \exp\left(-\frac{|y|}{\lambda}\right) dy}{\int_{\theta+1-x}^{+\infty} \frac{1}{2\lambda} \exp\left(-\frac{|y|}{\lambda}\right) dy}\right) \leq \frac{1}{\lambda}.$$

Recall that $\rho^{\top}(x) = 1/\lambda$ when $x < \theta + 1$. Therefore, $\rho(x) \leq \rho^{\top}(x)$ holds if $x < \theta + 1$.

Now consider that $x \geq \theta + 1$. In that case,

$$\rho(x) = \ln\left(\frac{1 - \frac{1}{2}\exp\left(\frac{\theta - x}{\lambda}\right)}{1 - \frac{1}{2}\exp\left(\frac{\theta + 1 - x}{\lambda}\right)}\right).$$

For convenience, we let $\alpha = \frac{1}{2}\exp\left(\frac{\theta+1-x}{\lambda}\right)$, and define a function f of α as follows:

$$f(\alpha) = \rho(x) - \frac{1}{\lambda}\exp\left(\frac{\theta + 1 - x}{\lambda}\right) = \ln\left(\frac{1 - \alpha e^{-1/\lambda}}{1 - \alpha}\right) - \frac{2\alpha}{\lambda}.$$

Observe that $\alpha \in (0, 1/2]$ whenever $x \geq \theta + 1$. Therefore, we can prove the lemma by showing that $f(\alpha) \leq 0$ for all $\alpha \in (0, 1/2]$. For this purpose, we first compute the second derivative of f with respect to α:

$$f''(\alpha) \quad = \quad (e^{1/\lambda} - 1) \cdot \frac{e^{1/\lambda} + 1 - 2\alpha}{(1-\alpha)^2 \cdot (e^{1/\lambda} - \alpha)^2}.$$

Given that $e^{1/\lambda} - 1 > 0$ and $e^{1/\lambda} + 1 - 2\alpha \geq e^{1/\lambda} > 0$, we have $f''(\alpha) > 0$. This indicates that

$$\max_{\alpha \in (0,1/2]} f(\alpha) = \max\{f(0), f(1/2)\}.$$

Meanwhile, $f(0) = 0$, and

$$\begin{aligned} f(1/2) &= \ln(2 - e^{-1/\lambda}) - \frac{1}{\lambda} \\ &= \ln\left(e^{1/\lambda} - \left(e^{1/2\lambda} - e^{-1/2\lambda}\right)^2\right) - \frac{1}{\lambda} \\ &< \ln(e^{1/\lambda}) - \frac{1}{\lambda} = 0. \end{aligned}$$

Therefore, $\max_{\alpha \in (0,1/2]} f(\alpha) \leq 0$, which proves the lemma. □

Proof of Theorem 4.1: The theorem directly follows from the privacy analysis in Section 4.2.3. □

Proof of Lemma 4.2: Let V be the set of all possible nodes in a quadtree built on D. We divide the nodes in V into three subsets: (i) the set V_1 of nodes that appear as non-leaf nodes in $\mathcal{T}^*$, (ii) the set V_2 of the nodes that appear as leaves in $\mathcal{T}^*$, and (iii) the set V_3 of the nodes that do not appear in $\mathcal{T}^*$. Let $g(S)$ be the expected number of nodes in a set S that appear in $\mathcal{T}$. Then, we have

$$\mathbb{E}[|\mathcal{T}|] \;=\; g(V_1) + g(V_2) + g(V_3) \;\leq\; |V_1| + |V_2| + g(V_3) \;=\; |\mathcal{T}^*| + g(V_3).$$

Therefore, the lemma can be proved by showing $g(V_3) \leq |\mathcal{T}^*|$.

Observe that each node in V_3 must be the descendant of a node in V_2, i.e., a node that appears as a leaf in $\mathcal{T}^*$. Therefore, we can divide the nodes in V_3 into $|V_2|$ subsets, such that all nodes in the same subset are descendants of the same node in V_2. Consider

any such subset S that corresponds to a node $v \in V_2$. Given that v appears as a leaf in $\mathcal{T}^*$, we have $c(v) \leq \theta$. In addition, since $|\mathcal{T}^*| > 1$, $\text{depth}(v) \geq 1$ holds. Therefore,

$$c(v) - \text{depth}(v) \cdot \delta \;\leq\; c(v) - \delta \;\leq\; \theta - \delta.$$

By Equation (4.7), we have $b(v) = \theta - \delta$. Furthermore, for any $v' \in S$, we have $c(v') \leq c(v)$, which also leads to $b(v') = \theta - \delta$. Therefore, v and v' have the same probability p_s to be split. Given $\delta = \lambda \cdot \ln \beta$, we have

$$\begin{aligned} p_s \;&=\; \Pr[\theta - \delta + \text{Lap}(\lambda) > \theta] \;=\; \Pr[\text{Lap}(\lambda) > \delta] \\ &=\; \int_{\lambda \cdot \ln \beta}^{\infty} \frac{1}{2\lambda} \exp\left(-\frac{|y|}{\lambda}\right) dy \;=\; \frac{1}{2\beta}. \end{aligned}$$

Assume that v appears in $\mathcal{T}$. Then, given that v is split with $\frac{1}{2\beta}$ probability, each child of v has $\frac{1}{2\beta}$ probability to appear in $\mathcal{T}$. Therefore, in expectation, the number of v's children that appear in $\mathcal{T}$ should equal $\beta \cdot \frac{1}{2\beta} = \frac{1}{2}$. In general, for any $i \geq 1$, there exist β^i nodes $v' \in S$ with $\text{depth}(v') - \text{depth}(v) = i$, and each such v' has $(\frac{1}{2\beta})^i$ probability to appear in $\mathcal{T}$. Hence, the expected number of nodes in S that appear in $\mathcal{T}$ is

$$g(S) \;=\; \sum_{i=1}^{+\infty} \beta^i \cdot \left(\frac{1}{2\beta}\right)^i \;=\; \sum_{i=1}^{+\infty} \frac{1}{2^i} \;=\; 1.$$

Since each S uniquely corresponds to a node in V_2, we have

$$g(V_3) \;=\; |V_2| \cdot g(S) \;= |V_2| \;\leq\; |\mathcal{T}^*|.$$

Therefore, the lemma is proved. □

Proof of Corollary 4.1: The corollary follows from Theorem 4.1 when $\gamma = \ln \beta$. □

Proof of Lemma 4.3: Let u and v be two nodes in $\mathcal{T}$, such that u is the parent of v. Then, $\text{hist}(v)[x] \leq \text{hist}(u)[x]$ holds for every symbol $x \in \mathcal{I} \cup \{\&\}$. Let x_v (resp. x_u) be the symbol that has the largest count in $\text{hist}(v)$ (resp. $\text{hist}(u)$). We have

$$\begin{aligned} c(v) \;&=\; \|\,\text{hist}(v)\|_1 - \text{hist}(v)[x_v] \;\leq\; \|\,\text{hist}(v)\|_1 - \text{hist}(v)[x_u] \\ &=\; \sum_{x \neq x_u} \text{hist}(v)[x] \;\leq\; \sum_{x \neq x_u} \text{hist}(u)[x] \\ &=\; \|\,\text{hist}(u)\|_1 - \text{hist}(u)[x_u] \;=\; c(u). \end{aligned}$$

Therefore, $c(\cdot)$ is monotonic. □

Proof of Theorem 4.2: Let D and D' be two neighboring datasets, such that D is obtained by inserting a sequence s into D'. Assume that $s = \$x_1 \dots x_l$, where $x_i \in I \cup \{\&\}$ for $i \in [1, l]$. To facilitate our proof, we define l datasets $D_1, D_2, \dots, D_l$, such that $D_i = D' \cup \{s_i\}$ and $s_i = \$x_1 x_2 \dots x_i$ is the length-i prefix of s ended at symbol x_i. Observe that $D_l = D$. For convenience, we define $D_0 = D'$.

In the following, we will prove that for any $i \in [1, l]$ and any output $\mathcal{T}$ of the modified PrivTree,

$$-\frac{\varepsilon}{l^\top} \;\leq\; \ln\left(\frac{\Pr[D_i \to \mathcal{T}]}{\Pr[D_{i-1} \to \mathcal{T}]}\right) \;\leq\; \frac{\varepsilon}{l^\top}, \tag{B.8}$$

where $\Pr[D_i \to \mathcal{T}]$ denotes the probability that PrivTree outputs $\mathcal{T}$ given D_i. This would prove the theorem because, given that $l \leq l^\top$,

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) \;=\; \sum_{i=1}^{l} \ln\left(\frac{\Pr[D_i \to \mathcal{T}]}{\Pr[D_{i-1} \to \mathcal{T}]}\right) \;\in\; [-\varepsilon, \varepsilon].$$

Observe that D_i can be obtained by appending a symbol x_i to the end of the sequence s_{i-1} in D_{i-1}. Therefore, when we change the input data from D_{i-1} to D_i, the only changes in the prediction suffix tree are the histogram counts that x_i contributes to. Observe that if x_i contributes to the prediction histogram $hist(v)$ of a node v, then $\mathrm{dom}(v)$ must be a suffix of s_{i-1}, and the only possible change in $\mathrm{hist}(v)$ is that $\mathrm{hist}(v)[x_i]$ would be increased by one. Then, by the definition of prediction suffix tree, all of those nodes v should form a path from the root to a leaf. In addition, by Equation (4.12), the score $c(v)$ of each of those nodes v is changed by at most one. In that case, we can prove Equation (B.8) by reusing the analysis in the proof of Theorem 4.1.

To explain, recall that the correctness of Theorem 4.1 only replies on two conditions. First, the score $c(v)$ of each node v is monotonic. Second, when we change the input data, all of the nodes affected should form a path from the root of the decomposition tree to a leaf, and the score of each of those nodes should change by at most one. Notice that all three conditions are satisfied when we change the input of the modified PrivTree from D_{i-1} to D_i. Combining this with the fact that the modified PrivTree uses a noise scale that is $l^\top$ times that of Algorithm 4.2, it can be verified that Equation (B.8) holds. Therefore, the theorem is proved. □

Proof of Theorem 4.3: Let D and D' be two neighboring datasets, such that D is obtained by inserting a sequence s into D'. Assume that $s = \$x_1 \dots x_l$, where $x_i \in I \cup \{\&\}$ for $i \in [1, l]$. Observe that each symbol x_i in s contributes to the prediction histograms of the nodes whose predictor strings $\mathrm{dom}(\cdot)$ are suffixes of $\$x_1 \dots x_{i-1}$. By the definition of prediction suffix tree, these nodes form a path from the root of $\mathcal{T}$ to a leaf. This indicates that each x_i gets counted in the histogram of *one* leaf node only. Taking into account $l \leq l^\top$, it follows that the sensitivity of releasing the histogram counts of all leaf nodes in $\mathcal{T}$ is $l^\top$. Therefore, adding i.i.d. Laplace noise $\mathrm{Lap}(\lambda)$ into each released count ensures ε-differential privacy if $\lambda \geq l^\top/\varepsilon$. □

Proof of Lemma 4.4: Consider three datasets $D_1 = \{a, b\}$, $D_2 = \{a, b, b\}$, and $D_3 = \{b, b\}$, where each tuple is either a or b. Observe that D_1 is a neighboring dataset of D_2, while D_2 is a neighboring dataset of D_3. Let q_a (resp. q_b) be a query that asks for the number of a (resp. b) in a dataset. Let Q be a sequence of k queries, such that first $k/2$ queries are all q_a, and the remaining $k/2$ queries are all q_b.

Suppose that we invoke Algorithm 4.4 on D_1, D_2, D_3, respectively, with Q, a noise scale λ, and a threshold $\theta = 1$. Let E be the event that Algorithm 4.4 outputs 1 for the first $k/2$ queries in Q, and 0 for the remaining $k/2$ queries. In addition, let $\Pr[D \to E]$ denote the probability that E occurs when the input dataset is D. If Algorithm 4.4 satisfies ε-differential privacy, then

$$\frac{\Pr[D_1 \to E]}{\Pr[D_2 \to E]} \leq e^{\varepsilon}, \qquad \frac{\Pr[D_2 \to E]}{\Pr[D_3 \to E]} \leq e^{\varepsilon}.$$

This indicates that

$$\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} = \frac{\Pr[D_1 \to E]}{\Pr[D_2 \to E]} \cdot \frac{\Pr[D_2 \to E]}{\Pr[D_3 \to E]} \leq e^{2\varepsilon}. \tag{B.9}$$

In what follows, we prove the lemma by showing that Equation (B.9) does not hold when $\lambda \leq k/4\varepsilon$.

Recall that Algorithm 4.4 generates a noisy threshold $\hat{\theta}$, and outputs 1 for a query q only when its noisy answer $\hat{q}(D)$ is larger than $\hat{\theta}$. Therefore,

$$\begin{aligned}\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} &= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\hat{q_a}(D_1) > x] \cdot \Pr[\hat{q_b}(D_1) \leq x]\big)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\hat{q_a}(D_3) > x] \cdot \Pr[\hat{q_b}(D_3) \leq x]\big)^{\frac{k}{2}} dx} \\ &= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) > x - 1] \cdot \Pr[\mathrm{Lap}(\lambda) \leq x - 1]\big)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\mathrm{Lap}(\lambda) > x] \cdot \Pr[\mathrm{Lap}(\lambda) \leq x - 2]\big)^{\frac{k}{2}} dx}.\end{aligned}$$

Consider any $x \in (-\infty, +\infty)$. If $x > 1$, then

$$\Pr[\text{Lap}(\lambda) > x - 1] \;=\; \frac{1}{2} e^{\frac{1-x}{\lambda}} \;=\; e^{\frac{1}{\lambda}} \cdot \Pr[\text{Lap}(\lambda) > x],$$

and $\Pr[\text{Lap}(\lambda) \leq x - 1] > \Pr[\text{Lap}(\lambda) \leq x - 2]$. This leads to

$$\begin{aligned} &\Pr[\text{Lap}(\lambda) > x - 1] \cdot \Pr[\text{Lap}(\lambda) \leq x - 1] \\ &\geq\; e^{\frac{1}{\lambda}} \cdot \Pr[\text{Lap}(\lambda) > x] \cdot \Pr[\text{Lap}(\lambda) \leq x - 2]. \end{aligned} \tag{B.10}$$

Meanwhile, if $x \leq 1$, then

$$\Pr[\text{Lap}(\lambda) \leq x - 1] \;=\; \frac{1}{2} e^{\frac{x-1}{\lambda}} \;=\; e^{\frac{1}{\lambda}} \cdot \Pr[\text{Lap}(\lambda) \leq x - 2],$$

and $\Pr[\text{Lap}(\lambda) > x - 1] > \Pr[\text{Lap}(\lambda) > x]$. In that case, Equation (B.10) still holds.

Given that Equation (B.10) holds for all $x \in (-\infty, \infty)$, we have

$$\begin{aligned} \frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} &= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\text{Lap}(\lambda) > x - 1] \cdot \Pr[\text{Lap}(\lambda) \leq x - 1]\big)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \big(\Pr[\text{Lap}(\lambda) > x] \cdot \Pr[\text{Lap}(\lambda) \leq x - 2]\big)^{\frac{k}{2}} dx} \\ &> \left(e^{\frac{1}{\lambda}}\right)^{\frac{k}{2}} = e^{\frac{k}{2\lambda}}. \end{aligned}$$

Therefore, $\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} > e^{2\varepsilon}$ when $\lambda \leq k/4\varepsilon$, which proves the lemma. □

Proof of Theorem 5.3: In what follows, we rewrite the proof of Theorem 6 of [McSherry and Talwar, 2007] using the concepts and notations developed in Chapter 5.

The first step is to observe that Algorithm 5.1 draws from a probability distribution over the integers where the probability associated with output k is proportional to $\exp(\varepsilon q(g, k)/2\Delta_q)$, normalized by the sum of this quantity over all integers. Note that we do not compute this probability directly; instead, for the sake of efficiency, we group together all integers which share the same probability (in our terminology, which are on the same rung) for the rungs 0 to M. Additionally, we compute the aggregate probability for all outputs on rungs below M as a single value captured by $M + 1$, as described after the description of the algorithm. It is straightforward to check that the actions of Algorithm 5.1 in the creation of the *weight* array give these sums of (unnormalized) probabilities. The action of the algorithm can then be seen as following two steps: first, we select a rung of the ladder (where rung $M + 1$ is considered as a special case) according to the relative value of the *weight* of the rung. Then, we pick an integer from the

corresponding rung. For rungs 0 to M, this is simple: all integers on the rung have the same probability, so we just pick one uniformly. For rung $M+1$, which corresponds to an unbounded tail of possible outputs, we take advantage of the structure to first pick how many further rungs down the ladder to go, and then to again pick uniformly from these. This gives the desired probability distribution of producing each possible value.

Hence as argued above, the output probability distribution of Algorithm 5.1 at integer k is equal to

$$\Pr[k] = \frac{\exp\left(\frac{\varepsilon}{2\Delta_q} \cdot q(g,k)\right)}{\sum_{k\in\mathbb{Z}} \exp\left(\frac{\varepsilon}{2\Delta_q} \cdot q(g,k)\right)}. \tag{B.11}$$

Now if the input g is replaced by its neighboring graph g', the quality of k will be changed by at most Δ_q, i.e., $|q(g,k) - q(g',k)| \leq \Delta_q$. Therefore, the numerator of (B.11) can change at most by a factor of $\exp(\varepsilon\Delta_q/2\Delta_q) = \exp(\varepsilon/2)$ and the denominator minimum by a factor of $\exp(-\varepsilon/2)$. Thus the ratio of the new probability of k (with g') and the original one is at most $\exp(\varepsilon)$. □

Proof of Theorem 5.5: We first prove that it is NP-complete to compute $LS_{ij}(g,t)$ given a particular pair of nodes (i,j), to which the well-known Clique Problem is polynomial-time reducible.

Given an instance of Clique Problem $\langle g,t\rangle$ (i.e., whether a given graph g has a clique of at least size t), we extend g to g^+ by adding two extra nodes i and j. Then we make i connected to all other nodes in g^+ while keeping j isolated (except i). This construction can be done in time that is polynomial in the size of Clique Problem instance. Next, we prove that

$$LS_{ij}(g^+,t) = \binom{t}{k-2} \quad \Leftrightarrow \quad g \text{ contains a } t\text{-clique},$$

where k can be set to any *constant* integer within $(3,t)$.

Recall from Lemma 5.4 that local sensitivity of $f_{k\mathbb{C}}$ over (i,j) is the number of $(k-2)$-cliques in the subgraph induced by their common neighbors A_{ij}. In g^+, A_{ij} is empty since j is isolated from g. After t modification steps on the graph, the size of A_{ij} could reach t if we connect j to t nodes in g (i is already connected to all nodes in g); and the maximum number of $(k-2)$-cliques, i.e., $\binom{t}{k-2}$, is achieved if and only if new neighbors of i and j form a t-clique. Therefore, it implies the existence of a t-clique in g.

We remark that this proof does not apply in the case of $k=3$ (triangle counting), since the t-clique is not the only subgraph of size t that maximizes the number of 1-cliques

(nodes). Indeed, computing $LS_{ij}(g,t)$ of triangle counting is polynomial-time solvable as shown by Lemma 5.2.

Last, we extend the result to $LS(g,t)$ through a reduction from the same NP-complete problem, Clique Problem. For any instance $\langle g,t\rangle$ where g has n nodes, we construct g^+ from g by adding a n-clique and a hub node i which is connected to all other nodes in g^+. Then we show that

$$LS(g^+,t) = \binom{n-1}{k-2} + \binom{t}{k-2} \quad \Leftrightarrow \quad g \text{ contains a } t\text{-clique.}$$

Let j be an arbitrary node in the n-clique. It is easy to prove that $LS(g^+,t) = LS_{ij}(g^+,t)$. Therefore, $LS(g^+,t)$ equals the number of $(k-2)$-cliques in the subgraph of A_{ij}. At the beginning, subgraph of A_{ij} is a $(n-1)$-clique. To extend it, one can use one modification to either (i) add a node of g to A_{ij} by connecting it to j, or (ii) add an edge into the subgraph of A_{ij}. The optimum is only attained by adding t nodes of g to A_{ij}, who form a t-clique in g. Thus, the new subgraph of A_{ij} contains two separated components: a $(n-1)$-clique and a t-clique, and it has $\binom{n-1}{k-2} + \binom{t}{k-2}$ different $(k-2)$-cliques. □

Proof of Lemma 4.5: Let D and D' be any pair of neighboring databases and $E = \{o_1, o_2, \ldots\}$ be any output returned by Algorithm 4.5. To prove the lemma, we shall show that $\frac{\Pr[D\to E]}{\Pr[D'\to E]}$ is always upper bounded by e^ε when $\lambda \geq 2/\varepsilon$.

Let $\mathbf{1}$ (resp. $\mathbf{0}$) denote the set of queries whose corresponding output in E equals 1 (resp. 0). As the algorithm terminates when the counter of the number of $o_i = 1$ reaches t, we know that the size of $\mathbf{1}$ should be no larger than t. Let $f(x)$ be the probability that the Laplace noise $\mathrm{Lap}(t\cdot\lambda)$ is larger than x, i.e., $f(x) = \Pr[\mathrm{Lap}(t\cdot\lambda) > x]$. Similarly, we define $g(x)$ as the probability of being no larger than x, that is, $g(x) = \Pr[\mathrm{Lap}(t\cdot\lambda) \leq x]$. Then, we can revise $\frac{\Pr[D\to E]}{\Pr[D'\to E]}$ as follows:

$$\begin{aligned}\frac{\Pr[D\to E]}{\Pr[D'\to E]} &= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta}=x]\cdot\prod_{q\in\mathbf{1}}\Pr[\hat{q}(D)>x]\cdot\prod_{q\in\mathbf{0}}\Pr[\hat{q}(D)\leq x]dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta}=x]\cdot\prod_{q\in\mathbf{1}}\Pr[\hat{q}(D')>x]\cdot\prod_{q\in\mathbf{0}}\Pr[\hat{q}(D')\leq x]dx} \\ &= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta}=x]\cdot\prod_{q\in\mathbf{1}} f(x-q(D))\cdot\prod_{q\in\mathbf{0}} g(x-q(D))dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta}=x]\cdot\prod_{q\in\mathbf{1}} f(x-q(D'))\cdot\prod_{q\in\mathbf{0}} g(x-q(D'))dx}. \end{aligned} \tag{B.12}$$

To figure out the upper bound of Equation (B.12), we first consider an easier case that D is obtained by adding a tuple into D'. This implies that $q(D') \leq q(D) \leq q(D')+1$ holds for any count query $q \in Q$. Given $q(D') \leq q(D)$ and the fact that $g(x)$ is a

monotonically increasing function of x, we have $g(x - q(D)) \leq g(x - q(D'))$. Moreover, we can remove the common factor $\Pr[\hat{\theta} = x]$ from both the numerator and denominator of (B.12). Thus, the equation is upper bounded by:

$$\frac{\Pr[D \to E]}{\Pr[D' \to E]} \leq \frac{\int_{-\infty}^{\infty} \prod_{q \in \mathbf{1}} f(x - q(D))dx}{\int_{-\infty}^{\infty} \prod_{q \in \mathbf{1}} f(x - q(D'))dx}.$$

Since $q(D) \leq q(D') + 1$ and $f(x)$ is monotonically decreasing on x, it is easy to have $f(x - q(D)) \leq f(x - q(D') - 1)$. Combined with the property of Laplace noise, we have

$$f(x - q(D)) \leq f(x - q(D') - 1) \leq e^{\frac{1}{t\lambda}} \cdot f(x - q(D')),$$

for any query $q \in \mathbf{1}$. Furthermore, given the fact that the size of $\mathbf{1}$ is at most t, (B.12) is upper bounded by $\exp(t \cdot \frac{1}{t\lambda}) = \exp(\frac{1}{\lambda})$.

Now we consider the other case that D' is obtained by inserting a tuple into D. As D' is a larger dataset, we have $q(D) \leq q(D') \leq q(D) + 1$. In this case, the $q \in \mathbf{0}$ terms could not be trivially eliminated, as $q(D)$ is no longer larger than $q(D')$. Fortunately, we manage to prove by utilizing the inequality $q(D') \leq q(D) + 1$. In particular, we shift the x-axis by 1 and replace x with $x + 1$ in the denominator of Equation (B.12):

$$\frac{\Pr[D \to E]}{\Pr[D' \to E]} = \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \prod_{q \in \mathbf{1}} f(x - q(D)) \cdot \prod_{q \in \mathbf{0}} g(x - q(D))dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x + 1] \cdot \prod_{q \in \mathbf{1}} f(x - q(D') + 1) \cdot \prod_{q \in \mathbf{0}} g(x - q(D') + 1)dx}.$$

Combining $q(D') - 1 \leq q(D)$ with the monotonicity of g, we again eliminate the $q \in \mathbf{0}$ terms from the upper bound. For other parts of the expression, we have

$$\Pr[\hat{\theta} = x] \leq e^{\frac{1}{\lambda}} \cdot \Pr[\hat{\theta} = x + 1],$$

and

$$f(x - q(D)) \leq f(x - q(D')) \leq e^{\frac{1}{t\lambda}} \cdot f(x - q(D') + 1)$$

for any query $q \in \mathbf{1}$. Provided that the size of $\mathbf{1}$ is at most t, the expression is upper bounded by $\exp(\frac{1}{\lambda} + t \cdot \frac{1}{t\lambda}) = \exp(\frac{2}{\lambda})$. Therefore, the lemma is proved. □

Appendix C

Publication List

- **PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions.**
  Jun Zhang, Xiaokui Xiao, and Xing Xie.
  Proceedings of the ACM International Conference on Management of Data (**SIGMOD**), to appear, 2016.

- **Private Release of Graph Statistics using Ladder Functions.**
  Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava and Xiaokui Xiao.
  Proceedings of the ACM International Conference on Management of Data (**SIGMOD**), pages 731-745, 2015.

- **PrivBayes: Private Data Release via Bayesian Networks.**
  Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava and Xiaokui Xiao.
  Proceedings of the ACM International Conference on Management of Data (**SIGMOD**), pages 1423-1434, 2014.

- **PrivGene: Differentially Private Model Fitting Using Genetic Algorithms.**
  Jun Zhang, Xiaokui Xiao, Yin Yang, Zhenjie Zhang, and Marianne Winslett.
  Proceedings of the ACM International Conference on Management of Data (**SIGMOD**), pages 665-676, 2013.

- **Functional Mechanism: Regression Analysis under Differential Privacy.**
  Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett.
  Proceedings of the 38th International Conference on Very Large Data Bases (**VLDB**), pages 1364-1375, 2012.