UNIVERSITY OF CALIFORNIA,
IRVINE

Towards Privacy Definition for Hybrid Sensitivity Data

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Stylianos Doudalis

Dissertation Committee:
Professor Sharad Mehrotra, Chair
Professor Chen Li
Associate Professor Stanislaw Jarecki

2017

DEDICATION

To my parents Dimitri and Elsa.
To my siblings Gianni, Thaleia.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

Cavafy, a Greek poet from Alexandria of Egypt, wrote in one of his most famous works: "*As you set out for Ithaca hope the voyage is a long one, full of adventure, full of discovery.*", "*Keep Ithaca always in your mind. Arriving there is what you are destined for. But do not hurry the journey at all. Better if it lasts for years, so you are old by the time you reach the island, wealthy with all you have gained on the way, not expecting Ithaca to make you rich.*" Ph.D. was a long journey for me, that helped me mature and broaden my horizons.

I would like to thank Prof. Sharad Mehrotra for guiding me through this long journey. He was always patient with me, and he offered me the opportunity to experiment with various topics. Without his ideas and his persistence, I would not have been able to complete my thesis.

I would like to offer my thanks to Prof. Ashwin Machanavajjhala and his student Ios Kotsogiannis for their cooperation, in order to complete the last piece of my thesis.

I am thankful to Prof. Chen Li, and Prof. Stanislaw Jarecki for being part of my dissertation committee.

I would like to express my gratitude to the ISG group, the Donald Bren School of ICS, and the University of California Irvine for creating a friendly and creative environment, where I could pursue my Ph.D.

On a personal level, I would like to thank all my friends, especially Blerim and Minas, for all fun moments that we had together.

Of course, I would not have been able to neither start nor complete my Ph.D. program without the support of my family. My brother Giannis, my sister Thaleia, and my father Dimitris were always on my side, when I needed them. Finally, my deepest appreciation and thanks to my mother, Elsa, for always encouraging me and believing in me.

CURRICULUM VITAE

Stylianos Doudalis

Doctor of Philosophy in Computer Science **2017**
University Of California Irvine *Irvine, California*

Diploma Degree in Computer Science **2010**
National Technical University of Athens *Greece*

Graduate Research Assistant **2013–2015**
University of California, Irvine *Irvine, California*

Teaching Assistant **2010-2013**
University Of California Irvine *Irvine, California*

PUBLICATIONS

"One-sided Privacy." Stylianos Doudalis, Ios Kotsogiannis, Samuel Haney, Ashwin Machanavajjhala, Sharad Mehrotra. Under Submission

"SORTaki: A Framework to Integrate Sorting with Differential Private Histogramming Algorithms." Stylianos Doudalis, Sharad Mehrotra. In: PST, 2017

"Releasing True Data with Formal Privacy Guarantees." Stylianos Doudalis, Samuel Haney, Ashwin Machanavajjhala, Sharad Mehrotra. In: PRIVACYPRESERVING IR, 2016

ABSTRACT OF THE DISSERTATION

Towards Privacy Definition for Hybrid Sensitivity Data

By

Stylianos Doudalis

Doctor of Philosophy in Computer Science

University of California, Irvine, 2017

Professor Sharad Mehrotra, Chair

Social media, web search logs and online purchases are only some of the sources used my private and public organizations to collect information about individuals. While, the aggregated data are valuable for research and commercial use, they can pose a direct threat to a user's privacy. The field of privacy preserving data sharing has emerged, in order to create tools, that can enable institutions and companies to shares their clients' information, while protecting their privacy.

Differential privacy (DP) has been recognized as the de-facto privacy framework for interacting with private information. In this thesis, we continue the work on release histograms with formal privacy guarantees. Particularly, we investigate the effect of sorting as a technique for improving the accuracy of the final approximation. We identify the right settings to use it, and when to avoid it.

DP aims to protect users' records from every possible type of inference attack. As a result, the DP is a very pessimistic and considers every piece of information as sensitive. In the context of this thesis, we will relax the previous requirement. We propose one-sided privacy (OSP) a novel privacy framework that is able to handle data that can be classified as sensitive or non-sensitive. Our empirical results show that OSP can support new types of application, and offers meaningful utility, in cases that DP is known to perform poorly.

Chapter 1

Introduction

User activity logs on search engines, e-commerce sites and social networks contain a wealth of information that has revolutionized search and advertising. These data are also valuable for furthering scientific and medical research (e.g. Google Flu [1]). However, since the release of AOL search logs and the Netflix challenge dataset in 2006, both of which led to widely publicized breaches of individual privacy, internet companies have become wary about releasing user activity data for research. Thus the field of privacy preserving data sharing was born, which tries to solve the complex problem of sharing user information while protecting the privacy of the individuals. The biggest challenge in this field has been on how to define privacy. Various privacy definitions have been proposed inspired from how an adversary could attack the user data.

Fast forward a decade and the field of anonymization and privacy preserving computation has modernized with a mathematical notion called ϵ-*differential privacy* (DP) [20] becoming the de facto standard for provably private data releases. An mechanism operating on a private database of records satisfies DP if its outputs are insensitive to the presence or absence of any single record in the input. The success of DP is due to three key properties. First, it

[1] www.google.org/flutrends

provides an attacker independent proof of privacy: a DP algorithm is provably insensitive to adding or removing a record from the database. This privacy guarantee is *future-proof* since an attacker's ability to learn sensitive information about an individual is bounded even if joined with new information about that individual in the future. Second, DP satisfies composition: composing multiple DP algoritihms, in parallel or sequentially, also results in a DP algorithms. Finally, DP has a *single* parameter ϵ that allows trading off privacy for utility. This has led to a slew of research in the academia [63, 66, 41, 16, 24, 29, 42, 60, 64, 33, 34, 18, 47, 68, 5, 13] and some real world adoption in government and commercial agencies [2].

In Chapter 3, as part of understanding DP, we created a technique for improving the quality of released histograms. Histograms is a fundamental tool in statistical analysis, machine learning and optimization of SQL queries. Traditionally, histograms function as summaries of the underlying data distribution. They offer a trade-off between accuracy and space. The goal of any histogramming technique is to create bins that are as uniform as possible, given the space limitation. The presence of a non-uniform bin means that there is an difference between the real frequencies of the values and their approximation by the bin's average frequency. The previous error is often referred to as generalization error. Had there been no space limitation, then the histogram would have a single bin per value and the generalization error would have been zero.

In the context of differential privacy, while space efficiency is still important, it is ignored in favor of reducing the noise added by differential privacy. The simplest DP technique releases a histogram by adding noise, sampled from a Laplace distribution, to the frequencies. The noise addition step introduced a new type of error, the perturbation error, which is comparable to the variance of the noise. Interestingly, the perturbation error is maximized if a histogram has multiple bins of size 1, and minimized when there is a single bin that engulfs all values,

[2] `https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/`

at the expense of maximizing the generalization error. Hence, in DP, the space constraint is replaced by the perturbation error and the process of creating a histogram needs to minimize the sum of the perturbation and generalization error.

Our contribution in improving histogram publication is the realization that, the bins in a histogram don't have to contiguous. The contiguous bins were originally used in order to satisfy the space constraint, since a bin can be represented using only two numbers. But, in the context of DP, the goal is to reduce the perturbation error and thus it makes sense to create clusters that contains values which similar frequencies. Like every Differential Private technique, this solution is best used for certain scenarios. It is suitable if the end goal is to answer small range queries, including queries of size 1.

DP follows a very pessimistic model, as it assumes that every piece of information is sensitive. In Chapter 4, we relax the previous assumption, and we explore a new path, where only a portion of the data is sensitive. Usually, classifying the data as sensitive or non-sensitive is not a easy task. But, in certain scenarios it can be naturally performed as a consequence of the laws, personal preferences and privacy rules.

In the context of our work, we use policies as a mechanism for specifying sensitivity. Query answering in the presence of policies has been traditionally part of access control literature. Unfortunately, access control solutions do not satisfy privacy, because they are susceptible to a specific type of attack, that we refer to as exclusion attack. All existing work in DP literature that considers multiple levels of privacy protection is vulnerable to exclusion attacks.

In this thesis, we formalize the exclusion attack and we propose a new privacy framework *one-sided privacy* (OSP). OSP offer differential privacy like guarantees, but only to the sensitive portion of the databases. OSP does not offer any protection for the non-sensitive records, as long as the privacy of sensitive data is not breached.

OSP has two very important side effects. First, it enable the release of a sample of non-sensitive record, while protecting against exclusion attacks. Additionally, it allows answering count queries over the non-sensitive data if $1/8$ of the error offers by DP. Our experiment show that OSP can offer great benefits in the context of complex analysis tasks, e.g. classification, and for releasing statistics over very large domains and with very high sensitivity.

The remain of this thesis is organized as follows. In Chapter 2, we introduce DP, its basic constructs and mathematical tools that we usually use in our proofs. In Chapter 3, we present our work on releasing histograms under DP. In Chapter 4, we present our novel privacy framework of OSP. Finally, we conclude this thesis and discuss future extensions in Chapter 5.

Chapter 2

Preliminaries

2.1 Differential Privacy

In this section, we present the fundamentals of Differential Privacy, i.e. the definition, the composition theorems and the basic constructs that can be used to achieve it. We conclude with a summary of privacy definitions that are inspired by Differential Privacy, but they go beyond the traditional case of relational databases.

2.1.1 Definition

Differential Privacy's definition is based on the notion of neighboring databases. Informally, if two databases differ on a single record, then they are neighbors. The previous informal statement is formally captured with the definitions of unbounded and bounded neighbors.

Definition 2.1 (Unbounded Neighbors [19])**.** *Let D, D' be two databases over domain $\mathcal{T}$. D, D' are neighbors, if and only if $|D - D' \cup D' - D| = 1$.*

Definition 2.2 (Bounded Neighbors [20])**.** *Let D, D' be two databases over domain $\mathcal{T}$. Let*

$r, r' \in \mathcal{T}$ be two records with $r \neq r'$. D, D' are neighbors, if and only if $D' = D - \{r\} \cup \{r'\}$.

Formally, Differential Privacy is defined as:

Definition 2.3 (Differential Privacy)**.** *An algorithm $\mathcal{M}$ satisfies ϵ-Differential Privacy, if and only if for every possible pair of neighbor databases D, D' and all $S \subseteq \mathcal{O}$:*

$$Pr[M(D) \in S] \leq e^{\epsilon} Pr[M(D') \in S]$$

It is important to note that Differential Privacy is a characteristic of algorithm $\mathcal{M}$, and it does not directly describe the statistical properties of the published data. Intuitively, Differential Privacy states that, if algorithm $\mathcal{M}$ is applied on two "similar" databases, i.e. neighbors, then the outputs of $\mathcal{M}$ will be "almost the same". Differential Privacy restricts the probability distribution of $\mathcal{M}$'s possible outputs. The probability of some output might be more probable in case of database D instead of D', but the ratio of the two probabilities is always bounded by a factor e^{ϵ}, where parameter ϵ is usually referred as privacy budget. Differential Privacy makes no assumptions about the adversary's background knowledge. Instead, it assumes a powerful adversary who knows every record in the database except for one. The definition of neighboring databases specifies the nature of the attack that an adversary is trying to launch based on the output of algorithm $\mathcal{M}$. In case of unbounded neighbors, the adversary's goal is to figure out if a record is present or absent in the database. According to bounded neighbors, the adversary is aware of a record's presence in the database, but s/he is trying to guess its true value. From a practical point of view, bounded neighbors offer a stricter version of Differential Privacy than unbounded neighbors. In [36], the authors provide an extended comparison of the two neighbor definition.

2.1.2 Composition Theorems

While the elegant mathematical definition, free of assumptions about adversarial knowledge, is considered the strongest point of Differential Privacy, the composition theorems are the second important contribution of Differential Privacy. The composition theorems offer an out of the box tool for arguing about privacy, when multiple differential private queries are answered over a database.

Theorem 2.1 (Sequential Composition)**.** *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be ϵ_1 and ϵ_2 differential private algorithms respectively. Then, $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ satisfies $(\epsilon_1 + \epsilon_2)$-One-sided Privacy.*

Sequential composition allows to reason about the privacy budget consumption, when multiple queries run on the "same" part (or overlapping parts) of a database. Intuitively, answering more queries "costs" more budget.

Theorem 2.2 (Parallel Composition Unbounded Neighbors)**.** *Let $\{D_1, \ldots, D_n\}$ be the partition of data D into disjoint parts. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be differential private mechanisms, with $\mathcal{M}_i$ running on D_i using ϵ_i budget. Then $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_n)$ satisfies $\max_i \epsilon_i$-One-sided Privacy.*

Theorem 2.3 (Parallel Composition Bounded Neighbors)**.** *Let $\{D_1, \ldots, D_n\}$ be the partition of data D into disjoint parts. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be differential private mechanisms, with $\mathcal{M}_i$ running on D_i using ϵ_i budget. Then $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_n)$ satisfies $\max_{i,j \neq i}(\epsilon_i + \epsilon_j)$-One-sided Privacy.*

Parallel composition considers the case that a database is "split" into disjoint parts and a single query is answered over each partition. For instance, imagine a medical database and a histogram about the number of patients per disease. Each database split contains all patients with the same disease and one query per split asks for the number of records in it. According to theorems 2.2, 2.3, since all queries run in "parallel", the total spent privacy

budget is $\max_i \epsilon_i$ ($\max_{i,j \neq i}(\epsilon_i + \epsilon_j)$) in case of unbounded (bounded) neighbors. In other word, if each query spent ϵ budget, then in case of unbound (bounded) neighbors the total budget spent is ϵ (2ϵ).

2.1.3 Basic Constructs

Differential Privacy provides basic constructs that can be used to develop more complex algorithms. In this subsection, we will overview Laplace [20, 22, 48] and Exponential [20, 22] mechanisms.

Laplace Mechanism

Laplace Mechanism was designed for publishing the output of continuous functions. On a high level, it releases a function's true output with noise sampled from a Laplace distribution. The noise is "large" enough, in order to hide the biggest possible change of the function's output between any two possible neighboring databases. The biggest possible change is formally defined as global sensitivity:

Definition 2.4 (Global Sensitivity)**.** *Given a function $f : \mathcal{D} \to \mathbb{R}$ and two neighboring databases D and D', the global sensitivity of f is $\triangle f = \max_{D,D'} \|f(D) - f(D)\|_1$.*

and Laplace Mechanism is defined as:

Definition 2.5 (Laplace Mechanism)**.** *For any function $f : \mathcal{D} \to \mathbb{R}$ and database D, mechanism $\mathcal{M}(D) = f(D) + Laplace(0, \triangle f/\epsilon)$, satisfies ϵ-Differential Privacy.*

Lets continue on the histogram example that we started in the previous subsection. We can answer the count queries required by the histogram using Laplace Mechanism. A count query has global sensitivity 1, because it can increase or decrease at most by one between any

two neighbor database. Therefore, we will add noise sampled from a Laplace distribution with scale $1/\epsilon$. If we combine the Laplace Mechanism with parallel composition, then we can learn the complete histogram using only ϵ privacy budget.

Exponential Mechanism

The Exponential Mechanism is a general technique that can be used to publish the output of arbitrary functions. Laplace Mechanism is a specialization of Exponential Mechanism for functions, whose range is a subset of real numbers. The key component of Exponential Mechanism is a utility function $u(D, r)$, that can provide a score for every possible response r for a given database D. Similarly to global sensitivity, the sensitivity of the utility method is the biggest possible change between any two neighbor databases:

$$\triangle u = \max_{r,D,D'} \|u(D, r) - u(D', r)\|_1$$

Exponential mechanism is defined as follows:

Definition 2.6 (Exponential Mechanism)**.** *Given a database D, a utility function $u(D, r) : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ and a privacy parameter ϵ, the mechanism*

$$M(D, u) = \textit{ choose } r \textit{ with probability proportional to} \exp(\frac{\epsilon \cdot u(D, r)}{2\triangle u})$$

satisfies ϵ-Differential Privacy.

Exponential Mechanism randomly selects a possible output with probability "proportional" to its utility score, such that responses with higher utility are more probable.

The Laplace and Exponential mechanisms are practical solutions only when the global sensitivity is small. For example, imagine a function that computes the average salary of the

doctors working in a hospital. The global sensitivity of the average salary is unbounded, thus Laplace Mechanism would add an "infinite" amount of noise. In [51] the authors proposed adding noise based on local sensitivity, i.e. the sensitivity computed based on a specific database instance. Of course, the real solution is not as simple as replacing the global with the local sensitivity, because that is not a differential private process.

2.2 Statistical/Mathematical Tools

In this section, we provide the basic statistical and mathematical background for better understanding of latter derivations.

2.2.1 Statistical Background

Expectation, Variance and Bias

Let X be a random variable with expected value (mean) $\mathbb{E}[X]$, its variance is defined as

$$\mathbb{V}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 \tag{2.1}$$

Let X, Y be two **independent** random variables, then the expectation of their product $Z = (aX)(bY)$, where a, b are constants, is

$$\mathbb{E}[Z] = ab\mathbb{E}[X]\,\mathbb{E}[Y]$$

Let X, Y be two **uncorrelated** random variables, then the expectation and variance of their

linear combination $Z = aX + bY + c$, where a, b, c are constants, are:

$$\mathbb{E}[Z] = a\mathbb{E}[X] + b\mathbb{E}[Y] + c$$
$$\mathbb{V}(Z) = a^2\mathbb{V}(X) + b^2\mathbb{V}(Y)$$

Let θ be an unknown parameter and $\hat{\theta}$ its estimator based on some observed data. The bias of estimator $\hat{\theta}$ is defined as:

$$Bias_\theta[\theta] = \mathbb{E}_\theta\left[\hat{\theta} - \theta\right] = \mathbb{E}_\theta\left[\hat{\theta}\right] - \theta$$

where the expectation is calculated over the randomness of the observed data. An estimator is said to be unbiased, if its bias is equal to zero for all values of parameter θ.

Exponential and Laplace Distributions

Exponential distribution with scale b has probability density function

$$f(x; b) = \begin{cases} \frac{1}{2}\exp(-\frac{x}{b}) & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Let X be a random variable that follows exponential distribution with scale b, then its expectation and variance are:

$$\mathbb{E}[X] = b$$
$$\mathbb{V}(X) = b^2$$

Laplace distribution, or often referred as double exponential distribution, with location μ and scale b has probability density function

$$f(x;\mu,b) = \frac{1}{2b}\exp(-\frac{|x-\mu|}{b})$$

Let X be a random variable that follows Laplace distribution with location μ and scale b, then its expectation and variance are:

$$\mathbb{E}[X] = \mu$$
$$\mathbb{V}(X) = 2b^2$$

2.2.2 Useful Mathematical Derivations

Decomposition of generalization error

In the context of histogram publication, generalization error appears as part of the error formula that estimates a histogram's error. Let $\mathbf{x}$ be a data vector with size n. If the histogram's error is defined using L_2^2 error metric, then the generalization error is equal to $\sum_{i=1}^{n}\left(x_i - \frac{\sum_{j=1}^{n} x_j}{n}\right)^2$ The cost for computing the generalization error is linear with the size of $\mathbf{x}$, i.e. $O(n)$. In [67], the authors show that it can be decomposed into the sum of the squared frequencies and the square of the sum of the frequencies as follows:

$$\sum_{i=1}^{n}\left(x_i - \frac{\sum_{j=1}^{n} x_j}{n}\right)^2 = \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n} \tag{2.2}$$

If the generalization error up to the $(n-1)^{th}$ element was computed and the intermediate results for terms $\sum_{i=1}^{n-1} x_i^2$ and $\sum_{i=1}^{n-1} x_i$ were saved, then we can compute the error for the whole

vector in constant time. From a practical point of view, equation 2.2 reduces the complexity of a histogramming construction techniques by one degree.

Chapter 3

Improving Histogram Accuracy through Sorting

3.1 Introduction

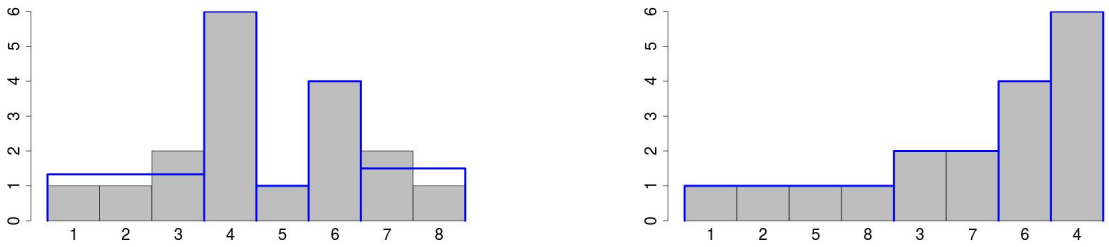
Today, personal information is being collected through various channels, such as, online surveys, electronic medical records, web search history, online transaction history etc. Sharing such microdata (individual specific data) can be of great scientific and commercial value at the cost of privacy violations. One powerful notion of privacy that has emerged to address these concerns is that of Differential Privacy (DP), first proposed by Dwork et al. [19, 20]. The DP criterion states that the output of a differentially private algorithm should not change significantly depending upon whether a single user's data are included in the input dataset or not. In other words, a DP-compliant algorithm should be "insensitive" to the presence or absence of a user's records in the input dataset. Given a user query, and a privacy budget, Differential Privacy perturbs the real answer by adding noise.

Histograms are one of the most convenient and easily understood ways of representing a

distribution and they are widely used for data analysis and data mining tasks. DP histogram publishing techniques can be separated into data-dependent [4, 12, 35, 67, 69] and data-independent [15, 28, 61] based on whether or not they make decisions that are influenced by the form of the distribution. Like traditional histogramming techniques [53], data-dependent algorithms consist of two phases. They begin by partitioning the domain into uniform bins and they represent each bin by the average frequency (or other similar metrics, e.g. median). The partitioning is based on a error measure that quantifies a bin's "goodness". Most of the times the error introduced due to a bin can be decomposed into two parts, the *generalization error* and the *perturbation error*. The former is the error created by grouping together values that have different real frequencies, while the latter is due to the noise added by Differential Privacy. The two types of errors have opposite behavior. If we create a single bin, we have the biggest possible generalization error, but we minimize the perturbation error. On the other extreme, if we assign a single bin per histogram value, then the generalization error is zero, but the perturbation error is maximized. It is this trade-off that all data dependent DP histogramming techniques exploit, in order to create better approximations of the underlying distribution compared to data independent techniques.

There are a variety of data-dependent algorithms that experiment with different partitioning algorithms. The partitioning algorithm may try to optimize histogram error based on L_1 or L_2^2 norm, it might approximate a bin's frequency using the average or median frequency. The partitioning algorithm might decide the bins' boundaries by working on noisy frequencies [62, 15, 67] or it might directly output them [12, 4, 67]. All this prior work has a common goal, that is to find a better way of quantifying a bin's error, in order to detect good regions of uniformity. In all cases the output is the same, a set of non-overlapping contiguous bins.

In [35, 69] the authors explore a new direction for improving the quality of data-dependent histogramming techniques. They observe that in the context of Differential Privacy we can go beyond contiguous bins. The continuous bins make sense, when the histogram acts as



(a) Original (b) Sorted

Figure 3.1: Sorting could potentially help to create bigger and with lower generalization error bins.

the summary of the underlying distribution. However, under Differential Privacy, while compactness of the outputted bins is a desirable property, the main focus is to reduce the error of the final noisy approximation. For example, consider the distribution in figure 3.1a that contains eight values. The overimposed blue line represents the histogram created by some data-dependent algorithm. Ideally, the algorithm should output uniform bins with zero generalization error. But, if we limit ourselves only to contiguous bins, then we will never consider clustering together values 1,2,5 and 8 because they are not "close" in the original domain. If we sort the values based on their frequency, i.e. figure 3.1b, then we could potentially create uniform bins. On a high level in [35, 69], sorting works as follows. A part of the budget is spent in order to learn the frequency of each value in the histogram. The noisy frequencies are used to re-order to domain in decreasing order. The partitioning and finalizing step are applied on the "sorted" domain that then the values are re-arranged to their original order.

Our contribution. In this paper, we continue and generalize the work started by [69]. Specifically, we make the following contributions.

- We built SORTaki, a general framework for incorporating sorting to any existing or future histograming technique. SORTaki has four basic components: 1) An *Initializer*, which collects some initial statistics about the underlying distribution. 2) a *Partitioner*, which represents any logic that can partition a domain into a set of contiguous bins, 3)

a *Finalizer*, which given a set of non-contiguous bins, creates the final approximation for each bin and 4) a *Sorter* which can re-order bins based on some criterion. SOR-Taki provides a flexible framework that we can use to easily experiment with various components and integrate sorting to existing histogramming algorithms like DPCube [62] and DAWA[12].

- We create a new partitioner and finalizer. In [69] a portion of the budget is used to decide the "sort" order and perform the partitioning. The information learned during the previous step is discarded and the final approximation is created using only the remainder of the privacy budget. In section 3.4, we first create a new *finalizer*, which can leverage the information collected during every step. Based on the new *finalizer*, we also derive a new bin error measure, which takes into account 1) the new *finalizer* and 2) the fact the generalization error is computed using noisy data. Our experiments show that the new *finalizer* and bin error measure can reduce the error up to 45%, for high levels of privacy budget or small domain sizes. Additionally, inspired by the partitioner presented in DAWA [12], we create a scalable dynamic programming based partitioner that offers up to 70 % improvement over the greedy partitioner presented in AHP [69].

- We perform a principled and thorough evaluation of old and new components based on the principles presented in DPBench [26], a general framework for evaluating differential private algorithms. Our goal is not only to evaluate the performance of existing and new techniques, but also to create general guidelines for using sorting based mechanisms, that could be potentially integrated into meta-algorithms like Pythia [39]. Based on our experiments the "ideal" distribution for sorting has a very big domain with hundred of thousand values, a very big scale (number records), the shape of distribution is not smooth, e.g. a power low distribution, and the workload consists of small range queries. Sorting best works in large domains, because it is easier to form

big non-contiguous bins. Large queries is a bad setting, because sorting creates non-contiguous bins that don't respect the ordering in the original domain. Therefore the bigger the range query is, the more likely is that it will be the union of multiple bins in the presence of sorting. Sorting actually tries to apply the *partitioner* and *finalizer* over a "smoothed" version of the distribution. Therefore, if the latter is already partially of fully "sorted", then what sorting tries to achieve comes for free. The smaller the scale of a dataset is the more likely is that there are big "regions" of zero counts in the distribution, which can be detected by any non-sorting based techniques.

The remainder of this chapter is organized as follows. Section 3.2 contains the related work. Section 3.3 describes SORTaki framework. Section 3.4 has the improved versions of partitioner and finalizer. Section 3.5 evaluates sorting enhanced algorithms and identifies the right setting for sorting.

3.2 Related Work

Publishing histograms in the context of differential privacy is a well studied problem. In this section we classify the majority of algorithms based on a variety of criteria.

DP histogramming techniques can be classified as data dependent vs data independent. The major advantage of **data independent** [4, 15, 28, 54, 61] technique is their stability, as they have nice foreknown behavior. Stability, however, comes at the cost of performing poorly in case of small range queries and not exploiting the possible inherent uniformity that might exists in the dataset. All **data dependent** algorithms are two phase. The algorithms start with a bin construction phase, which partitions the domain into disjoint bins, followed by a querying phase that learns the frequencies of each bin. Different approaches have been proposed for both phases. In [4, 67] the authors publish the bins using the exponential

mechanism. In [67] the authors create the bins based on noisy frequencies using a dynamic programing approach, while in [15, 62] they employ a Kd-tree approach for partitioning multidimensional data. In [12], the binning algorithm selects the optimal scheme by adding noise directly to candidate bin's error (generalization plus perturbation error). The usual approach for a second phase is to query the bin's average (or median) frequency. Instead, in [67] the authors use the boost technique [28], which improved the algorithm's stability and its performance on range queries.

DP histogramming methods can also be classified based on additional knowledge used by the algorithm. In [12, 25, 43] the authors present **workload aware** histogramming techniques using both data dependent and data independent approaches. Also in [25] the algorithm can be classified as following the interactive setting where the user is not necessarily interested in the complete distribution. Orthogonal to the work of publishing histograms is that of learning **k-way marginal contingency** tables [55].

Sorting has been explored in [35, 69]. In [35] the authors exploit sorting for the specific scenario of publishing histograms when the database contains multiple records of a user. They tackle the problem by publishing a histogram with fixed size bins. Their approach exploits part of the privacy budget to sample the data distribution, in order to sort the domain based on frequencies and to learn the optimal bin size. In [35] the authors show, that the resulting data dependent approach outperforms the data independent strategy of publishing a DP histogram with fixed size bins in the original domain order. While the approach in [35] is related to us in motivation, our goals are different. Instead of the specific scenario explored in [35], we explore a general mechanism to incorporate sorting into any data dependent algorithm and investigate, in depth, the conditions under which such an approach may improve utility. Our work can be considered as a continuation and generalization of the work started in [69].

Given the plethora of the proposed algorithms and the fact that there is no single best

solution, a new line of work [26, 39] tries to impose order in the "chaos". In [26], the authors present DPBench, a framework for evaluating differiential private histogramming solutions. Our experimental setup in section 3.5.1 is based on the principles of DPBench. In [39], Ios et at. created a meta-algorithm, which enables the user to select the right histogramming mechanism. The conclusions from on our empirical evaluation can help improve such meta-algorithm on how to use sorting based techniques.

3.3 SORTaki framework

In order to extend data dependent algorithms with sorting, we first describe an abstract/conceptual view of such algorithms which will allow us to incorporate different strategies of sorting into the existing algorithms.

3.3.1 Abstract Components

While individual details differ, we can intuitively view all data-dependent algorithms as consisting of three major components – an *initializer* that might create a noisy representation of the original histogram, a *partitioner* that splits the domain into a set of not overlapping bins, and a *finalizer*, that, given the bins, creates the final histogram. The privacy budget is split between the three components, in the ration γ_{in}, γ_p and $1 - \gamma_{in} - \gamma_p$, where the overall privacy budget is ϵ. We first describe the three components in more detail and then show how this abstraction can be instantiated to realize two popular DP algorithms – DAWA [12] and DPCube [62].

Symbol	Explanation
$N(b)$	Random variable sampled from Laplace distribution with scale b.
v / $\|v\|$ / $\mathcal{V}$	Bin / Bin's size / Set of bins
γ_{in} / γ_p / γ_f	Portion of budget used by initializer/partitioners/finalizer.
ϵ	Total privacy budget.
ϵ_{in} / ϵ_p / ϵ_f	$\gamma_{in}\epsilon$ / $\gamma_p\epsilon$ / $\gamma_f\epsilon$
x_i / $\widetilde{x}_{i,\epsilon}$	Ith value's real frequency / $x_i + N_i(1/\epsilon)$
$\mathcal{H}$ / $\widetilde{\mathcal{H}}$ / $\widetilde{\mathcal{H}_f}$	original histogram / initializer's output / finalizer's output.
$\mathcal{T}$	Optional module parameters.
$Identity(\|\mathcal{H}\|)$	Creates one bin per value
avg_v / $\widetilde{avg}_{v,\epsilon}$	v's real average frequency / $avg_v + N_v(\frac{1}{\|v\|\epsilon})$
$\widehat{avg}_v$	v's final average noisy frequency.
$\mathbb{E}[X]$	Expected value of random variable X
$\mathbb{V}[X]$	Variance of random variable X
$EE(v)$	v's expected error
$UEE(v)$	v's unbiased expected error

Table 3.1: Notation Summary

Initializer

The *initializer* optionally prepares the original histogram $\widetilde{\mathcal{H}}$ for the *partitioner* module. For example, DPCube's *partitioner* makes the decisions over noisy frequencies, while DAWA's *partitioner* requires access to the real data. A basic version of the *initializer* is described in algorithm 1. If γ_{in} is greater than zero, e.g. DPCube, then the module will output a noisy

Algorithm 1 Basic Initializer: $\mathcal{H}, \epsilon, \gamma_{in}, \mathcal{T}$

$\widetilde{\mathcal{H}} = H, \epsilon_{in} = \gamma_{in}\epsilon$
if $\gamma_{in} > 0$ **then**
  $\widetilde{\mathcal{H}} = (\widetilde{x}_{1,\epsilon_{in}}, \ldots \widetilde{x}_{|H|,\epsilon_{in}})$, s.t. $\widetilde{x}_{i,\epsilon_{in}} = x_i + \tilde{L}(0, 1/\epsilon_{in})$
  Return $\widetilde{\mathcal{H}}$
end if
Return [].

version of $\mathcal{H}$, using the privacy budget $\gamma_{in}\epsilon$. Otherwise, it returns an empty array. Argument $\mathcal{T}$ represents a set of extra parameter that might be unique to custom *initalizers*. Thus, we

propose the following template for the *initializer* module:

$$\widetilde{\mathcal{H}} = Initializer(\mathcal{H}, \epsilon, \gamma_{in}, \mathcal{T})$$

Partitioner

The *partitioner* is a wrapper around the logic that splits the original domain into contiguous bins. The input to the *partitioner* is a set of bins $\mathcal{V}$. Initially, all the bins inside $\mathcal{V}$ have size equal to 1. Using the information contained in $\mathcal{H}$ or $\widetilde{\mathcal{H}}$, the *partitioner* decides how to merge the smaller bins to larger ones. The binning process spends γ_p portion of the total budget ϵ. Usually, the *partitioner* needs to make assumptions about how the *initializer* and the *finalizer* work. Therefore, it requires to know their portion of the budget, i.e. γ_{in} and γ_p. Finally, $\mathcal{T}$ represents a set of parameters that are unique to every *partitioner* algorithm, such as thresholds or parameter controlling the behavior of internal data structures, e.g. the fanout of a tree. The previous description is summarized into the following template:

$$\mathcal{V} = Partitioner(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$$

Finalizer

The finalizer creates the final differentially private histogram $\widetilde{\mathcal{H}_f}$ using the remainder of the budget, i.e. $\gamma_f\epsilon$, and $\widetilde{\mathcal{H}}$. Given a set of bins, a *finalizer* employees a query strategy and it also leverages post processing techniques, e.g. least square fit, in order to improve the utility. The query strategy, i.e. the set of queries asked using γ_f, might be pre-defined, e.g.

asking a single query per bin, or workload aware. The *finalizer* has similar template to the *partitioner*.

$$\widetilde{\mathcal{H}_f} = Finalizer(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$$

Mapping existing algorithms to the abstraction layer

Using the *initializer*, the *partitioner* and the *finalizer* modules, it is straightforward to describe data-dependent histogramming algorithms as presented in algorithm 2. The initializer spends budget ϵ_{in} to generate a noisy histogram using Laplace mechanism. $\mathcal{V}$ is initialized into a set of bins that contains a single value per bin. The *partitioner* decides the final bins and the finalizer creates the output histogram $\widetilde{\mathcal{H}_f}$.

Algorithm 2 Algorithm Template without Sorting: $\mathcal{H}, \epsilon, Initializer, \gamma_{in}, Partitioner, \gamma_p, Finalizer$

$\widetilde{\mathcal{H}} = Initializer(\mathcal{H}, \gamma_{in}, \mathcal{T})$
2: $\mathcal{V} = Identity(|H|)$
$\mathcal{V} = Partitioner(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$
4: $\widetilde{\mathcal{H}_f} = Finalizer(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$
Return $\widetilde{\mathcal{H}_f}$

Next, we illustrate how the above abstract formulation of data-dependent algorithms can be instantiated to realize two well known algorithms –viz, DAWA [12] and DPCube [62].

DAWA. DAWA does not contain an explicit initializer, i.e. $\gamma_{in} = 0$. Instead, DAWA's *partioner* decides the bins while looking at the real data. It considers all possible contiguous bins as possible candidates. For each bin, it calculates its error based on the real data and the error is perturbed. The noise addition guarantees that the bin selection is a differentially private process. A deterministic dynamic programming algorithm selects the best set of non

overlapping contiguous bins based on the noisy bins' errors. DAWA's *partitioner* has an optional parameter that allows to select only candidate bins of size multiple of 2. DAWA's *finalizer* employs a tree based query strategy. For example imagine that $\mathcal{H}$ was split into two bins v_1 and v_2. The query strategy consists of 3 candidate query, one for each bin and one for for their union. For each query it allocates a privacy budget based on a workload. For example, if the workload consists of small queries, then it will decide to ask a single query per bin. On the other extreme, if the workload contains a lot of large ranges, it will favor queries that span multiple bins. The extra parameters $\mathcal{T}$ for the finalizer consist of the workload and the fanout of the tree.

DPCube. DPCube starts by creating a noisy version of $\mathcal{H}$, i.e. $\gamma_{in} > 0$. The partitioner does not spend additional budget, i.e. $\gamma_p = 0$. DPCube's *partitioner* uses a KD-Tree like algorithm to split the domain into bins. A tree node is split, if its noisy variance is bigger than a threshold ξ. The finalizer asks a single query (average) for each bin, followed by a post-processing step that combines the average stats with the information inside $\widetilde{\mathcal{H}}$.

3.3.2 Adding Sorting

In the previous subsection, we saw how techniques like DAWA and DPCube can be implemented in our framework. In this subsection, we extend algorithm 2 to support sorting based solution and we use AHP [69] as an example.

Bin as a subset of the domain

Given that sorting does not respect the original domain order, we extend the definition of bin, v, to a subset of the domain. Going back to the introductory example in figure 3.1a, every bin corresponds to a single range (or contiguous bin). On the other hand, after sorting

(figure 3.1b), a bin can be any subset of the original domain. The complete partitioning, $\mathcal{V}$, is constrained such that: (a) no two bins overlap, and (b) union of the bins covers the entire domain. We will, henceforth, refer to such generalized bins, simply as bins.

Sorter

A *sorter* takes as input a set of bins $\mathcal{V}$ and a noisy histogram $\widetilde{\mathcal{H}}$ and generates as output the input bins sorted based on some criterion, e.g. the average bin frequency.

$$\mathcal{V} = Sorter(\mathcal{V}, \widetilde{\mathcal{H}})$$

Algorithm 3 Algorithm Template with Sorting: $\mathcal{H}, \epsilon, Initializer, \gamma_{in} > 0$, $Partitioner, \gamma_p, Finalizer$

$\widetilde{\mathcal{H}} = Initializer(\mathcal{H}, \gamma_{in}, \mathcal{T})$
2: $\mathcal{V} = Identity(|H|)$
$\mathcal{V} = Sorter(\mathcal{V}, \widetilde{\mathcal{H}})$
4: $\mathcal{V} = Partitioner(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$
$\widetilde{\mathcal{H}_f} = Finalizer(\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p, \gamma_f, \mathcal{T})$
6: Return $\widetilde{\mathcal{H}_f}$

The complete logic of a data-dependent algorithm enhanced with sorting is presented in algorithm 3. The main change is that the partitioner is preceded by the *sorter*, that re-orders the bins. The *partitioner* and *finalizer* work on the "sorted" domain the same way that they would in the original domain, but the *finalizer* is additionally required to create final approximation based on the original domain order. Since sorting has to be done in a differental private fashion, it is always performed on noisy frequencies, thus parameter γ_{in} is required to be greater than 0.

Implementing AHP

AHP [69] can be easily implemented based on algorithm 3. AHP's spends part of the budget in order to learn a noisy version of the distribution like in the basic intitializer (algorithm 1), but additionally is employs a post-processing step which map every frequency smaller that a threshold ξ to zero. We implemented the thresholding step as an extension of the basic initializer. After sorting the domain, AHP created bins based on $\widetilde{\mathcal{H}}$ without spending extra budget, i.e. $\gamma_p = 0$. AHP's finalizer consists of asking for every bin's average frequency using the remainder budget.

Above, we showed how AHP algo can be implemented using our framework. In a similar manner, algorithsm such as DAWA and DPCube can be augmented to exploit sorting as well, by adding a step between initialization and partitioning to sort the domain. Indeed, we implemented several of these algorithms and compare them with and without sorting in section 3.5.2.

3.4 New Components

In the previous section, we presented our framework and we showed how DAWA, DPCube and AHP can be instantiated. In this section, we suggest new components based on the shortcomings of previous work or the new ideas applied in the literature in different settings.

Our first contribution is the WA *finalizer* . AHP's *finalizer* is simple. Each bin's frequency is approximated by the average frequency learned using the remainder of privacy budget, i.e. $\gamma_f\epsilon$ and any information present in $\widetilde{\mathcal{H}}$, i.e. the output of the *initializer*, is ignored. In this section, we create the WA *finalizer*, that is the weighted average of the average that can be learned from $\widetilde{\mathcal{H}}$ the average directly queries from the database using the last part of the privacy budget. DPCube has a similar *finalizer*, that combines the two sources of

information with a least square technique, but we provide a closed form solution. We go one step more and we create a *partitioner* based on WA *finalizer*. In subsection 3.4.2, we derived it bin error formula. Our second extension is an modular greedy and dynamic programming based *partitioners* that can partition the domain using any bin error formula.

3.4.1 Weighted Average Finalizer (WAF)

In this subsection, we derive the formula for our new *finalizer*, that can leverage all the statistical information learned through all the steps of histogram construction. Imagine that a *partitioner* somehow decided the final bins $\mathcal{V}$. What should be the final frequency for each value inside $v \in \mathcal{V}$?

$$\widehat{avg}_v = \alpha_1 \frac{1}{|v|} \sum_{i \in v} \widetilde{x}_{i,\epsilon_{in}} + \alpha_2 \cdot \widetilde{avg}_{v,\epsilon_f} \tag{3.1}$$

For that task we estimate bin's final frequency, i.e. $\widehat{avg}_v$, as the combination of two estimators. The first one, i.e. $\frac{1}{|v|} \sum_{i \in v} \widetilde{x}_{i,\epsilon_{in}}$, computes the average using $\widetilde{\mathcal{H}}$, while the second one, i.e. $\widetilde{avg}_{v,\epsilon_f}$, is learned from the database using the remainder of the budget. Next, we derive weights, α_1 and α_2, such that $\widehat{avg}_v$ is unbiased and it has minimum variance. Let $N_{i,\epsilon_{in}}$ and N_{v,ϵ_f} be Laplace random variables with scales $1/\epsilon_{in}$ and $1/(|v|\epsilon_f)$ respectively. The estimator's bias is expectation

$$\begin{aligned} Bias[\widehat{avg}_v] = 0 \Leftrightarrow \\ \mathbb{E}\left[\widehat{avg}_v - avg_v\right] = 0 \Leftrightarrow \\ \mathbb{E}\left[\alpha_1 \frac{1}{|v|} \sum_{i \in v} \widetilde{x}_{i,\epsilon_{in}} + \alpha_2 \cdot \widetilde{avg}_{v,\epsilon_f} - avg_v\right] = 0 \Leftrightarrow \end{aligned}$$

$$\mathbb{E}\left[\alpha_1 \frac{1}{|v|}\sum_{i\in v}(x_i + N_{i,\epsilon_{in}}) + \alpha_2 \cdot (avg_v + N_{v,\epsilon_f}) - avg_v\right] = 0 \Leftrightarrow$$

$$\mathbb{E}\left[\alpha_1 \frac{1}{|v|}\sum_{i\in v} x_i + \alpha_2 \cdot avg_v - avg_v + \alpha_1 \frac{1}{|v|}\sum_{i\in v} N_{i,\epsilon_{in}} + \alpha_2 \cdot N_{v,\epsilon_f}\right] = 0 \Leftrightarrow$$

$$\mathbb{E}\left[(\alpha_1 + \alpha_2 - 1)\cdot avg_v + \alpha_1 \frac{1}{|v|}\sum_{i\in v}\mathbb{E}\left[N_{i,\epsilon_{in}}\right] + \alpha_2 \cdot \mathbb{E}\left[N_{v,\epsilon_f}\right]\right] = 0 \Leftrightarrow$$

The average of noise is 0, i.e. $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = \mathbb{E}\left[N_{v,\epsilon_f}\right] = 0$

$$(\alpha_1 + \alpha_2 - 1)\cdot avg_v = 0 \tag{3.2}$$

Since we imposed the restriction that the estimator is unbiased it trivially holds that

$$\mathbb{E}\left[\widehat{avg}_v\right] = avg_v \tag{3.3}$$

The estimator's variance is

$$\mathbb{V}\left(\widehat{avg}_v\right) = \mathbb{E}\left[(\widehat{avg}_v - \mathbb{E}\left[\widehat{avg}_v\right])^2\right]$$

Substituting equation (3.3)

$$= \mathbb{E}\left[(\widehat{avg}_v - avg_v)^2\right]$$

$$= \mathbb{E}\left[\left(\alpha_1 \frac{1}{|v|}\sum_{i\in v}\widetilde{x}_{i,\epsilon_{in}} + \alpha_2 \cdot \widetilde{avg}_{v,\epsilon_f} - avg_v\right)^2\right]$$

$$= \mathbb{E}\left[\left(\alpha_1 \frac{1}{|v|}\sum_{i\in v}(x_i + N_{i,\epsilon_{in}}) + \alpha_2 \cdot (avg_v + N_{v,\epsilon_f}) - avg_v\right)^2\right]$$

$$= \mathbb{E}\left[\left((\alpha_1 + \alpha_2 - 1)\cdot avg_v + \alpha_1 \frac{1}{|v|}\sum_{i\in v} N_{i,\epsilon_{in}} + \alpha_2 \cdot N_{v,\epsilon_f} - avg_v\right)^2\right]$$

Using equation (3.2)

$$= \mathbb{E}\left[\left(\alpha_1 \frac{1}{|v|}\sum_{i\in v} N_{i,\epsilon_{in}} + \alpha_2 \cdot N_{v,\epsilon_f}\right)^2\right]$$

$$= \frac{\alpha_1^2}{|v|^2} \sum_{i=1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}^2\right] + \alpha_2^2 \mathbb{E}\left[N_{v,\epsilon_f}^2\right] + \frac{2\alpha_1\alpha_2}{|v|} \sum_{i=1}^{|v|} \mathbb{E}\left[N_{v,\epsilon_f} N_{i,\epsilon_{in}}\right] +$$

$$+ 2\alpha_1^2 \sum_{i=1}^{|v|} \sum_{j=i+1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}} N_{j,\epsilon_{in}}\right]$$

N_{v,ϵ_f} is independent of $N_{i,\epsilon_{in}}$ and $N_{i,\epsilon_{in}}$ is independent of $N_{j,\epsilon_{in}}$ for $i \neq j$

$$= \frac{\alpha_1^2}{|v|^2} \sum_{i=1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}^2\right] + \alpha_2^2 \mathbb{E}\left[N_{v,\epsilon_f}^2\right] + \frac{2\alpha_1\alpha_2}{|v|} \mathbb{E}\left[N_{v,\epsilon_f}\right] \sum_{i=1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}\right] +$$

$$+ 2\alpha_1^2 \sum_{i=1}^{|v|} \sum_{j=i+1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}\right] \mathbb{E}\left[N_{j,\epsilon_{in}}\right]$$

The average of noise is 0, i.e. $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = \mathbb{E}\left[N_{j,\epsilon_{in}}\right] = \mathbb{E}\left[N_{v,\epsilon_f}\right] = 0$

$$= \frac{\alpha_1^2}{|v|^2} \sum_{i=1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}^2\right] + \alpha_2^2 \mathbb{E}\left[N_{v,\epsilon_f}^2\right]$$

Since $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = \mathbb{E}\left[N_{v,\epsilon_f}\right] = 0$ from equation 2.1, we get $\mathbb{V}\left(N_{i,\epsilon_{in}}\right) = \mathbb{E}\left[N_{i,\epsilon_{in}}^2\right]$ and $\mathbb{V}\left(N_{v,\epsilon_f}\right) = \mathbb{E}\left[N_{v,\epsilon_f}^2\right]$

$$= \frac{\alpha_1^2}{|v|^2} \sum_{i=1}^{|v|} \mathbb{V}\left(N_{i,\epsilon_{in}}\right) + \alpha_2^2 \mathbb{V}\left(N_{v,\epsilon_f}\right)$$

$$= \frac{\alpha_1^2}{|v|^2} \sum_{i=1}^{|v|} \frac{2}{\epsilon_{in}^2} + \alpha_2^2 \frac{2}{|v|^2 \epsilon_f^2}$$

$$= \frac{2\alpha_1^2}{|v|\epsilon_{in}^2} + \frac{2(1-\alpha_1)^2}{|v|^2 \epsilon_f^2} \tag{3.4}$$

Next, we compute the minimum of quadratic equation 3.4 with respect to parameter α_1. Hence, we compute the value of parameter α_1 that renders the first derivative of the variance with respect to parameter α_1 equal to zero.

$$\frac{\partial \mathbb{V}\left(\widehat{avg}_v\right)}{\partial \alpha_1} = 0 \Leftrightarrow$$

$$\frac{4\alpha_1}{|v|\epsilon_{in}^2} + \frac{4(1-\alpha_1)}{|v|^2 \epsilon_f^2}(-1) = 0 \Leftrightarrow$$

$$\frac{4}{|v|^2 \epsilon_{in}^2 \epsilon_f^2} \left(|v|\epsilon_f^2 \alpha_1 + \epsilon_{in}^2(\alpha_1 - 1)\right) = 0 \Leftrightarrow$$

$$\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right) \alpha_1 - \epsilon_{in}^2 = 0 \Leftrightarrow$$

$$\alpha_1 = \frac{\epsilon_{in}^2}{\epsilon_{in}^2 + |v|\epsilon_f^2} \tag{3.5}$$

The first derivative is smaller than 0 on the left of $\frac{\epsilon_{in}^2}{\epsilon_{in}^2+|v|\epsilon_f^2}$ and bigger than 0 on the right of $\frac{\epsilon_{in}^2}{\epsilon_{in}^2+|v|\epsilon_f^2}$, therefore $\frac{\epsilon_{in}^2}{\epsilon_{in}^2+|v|\epsilon_f^2}$ is the minimum.

Since, $\alpha_2 = 1 - \alpha_1$, we have that

$$\alpha_2 = \frac{|v|\epsilon_f^2}{\epsilon_{in}^2 + |v|\epsilon_f^2} \tag{3.6}$$

Based on equations (3.5), (3.6), we can derive when $\widetilde{avg}_{v,\epsilon_f}$ is a better choice than using the information in $\widetilde{\mathcal{H}}$:

$$\alpha_2 > \alpha_1 \Leftrightarrow \sqrt{|v|}\epsilon_f > \epsilon_{in} \tag{3.7}$$

Based on equation (3.7), $\widetilde{avg}_{v,\epsilon_f}$, i.e. the average learned directly using privacy budget ϵ_f, is a better that the estimator based on $\widetilde{\mathcal{H}}$ in two cases: 1) ϵ_f is large enough or 2) the bin's size is big. At this point, using equations (3.5), (3.6), we can compute the final frequency for each bin based on equation 3.1.

Next we replace the value of equation (3.5), in equation 3.4, in order to compute the final closed form for a bin's variance, which will be useful in the following subsection.

$$\mathbb{V}\left(\widehat{avg}_v\right) = \frac{2\alpha_1^2}{|v|\epsilon_{in}^2} + \frac{2(1-\alpha_1)^2}{|v|^2\epsilon_f^2}$$

Substituting equation (3.5)

$$= \frac{2\epsilon_{in}^4}{|v|\epsilon_{in}^2\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)^2} + \frac{2|v|^2\epsilon_f^4}{|v|^2\epsilon_f^2\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)^2}$$

$$= \frac{2\epsilon_{in}^2}{|v|\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)^2} + \frac{2\epsilon_f^2}{\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)^2}$$

$$= \frac{2\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)}{|v|\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)^2}$$
$$= \frac{2}{|v|\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)} \tag{3.8}$$

In this subsection, we derived the closed form formula for the WAF *finalizer* and we explained how the weights in the formula affect the final estimation of a bin's formula.

3.4.2 WA based bin error estimation formula

In this subsection, we derive the error formula, that evaluates a bin's quality, assuming WA *finalizer*. The first step towards that direction is to measure how good of an approximation is $\widehat{avg}_v$ for v, in the ideal scenario that we had access to v's real frequencies. We measure the quality with the expected sum of squared differences between f_i and $\widehat{avg}_v$ or $EE(v)$ for short.

$$EE(v) = \mathbb{E}\left[\sum_{i=1}^{|v|}(x_i - \widehat{avg}_v)^2\right]$$
$$= \sum_{i=1}^{|v|}\mathbb{E}\left[(x_i - \widehat{avg}_v)^2\right] =$$
$$= \sum_{i=1}^{|v|}\mathbb{E}\left[(x_i - avg_v + avg_v - \widehat{avg}_v)^2\right]$$
$$= \sum_{i=1}^{|v|}(x_i - avg_v)^2 + \sum_{i=1}^{|v|}\mathbb{E}\left[(avg_v - \widehat{avg}_v)^2\right] + \sum_{i=1}^{|v|}2(x_i - avg_v)\mathbb{E}\left[avg_v - \widehat{avg}_v\right]$$

$\widehat{avg}_v$ is unbiased thus $\mathbb{E}\left[avg_v - \widehat{avg}_v\right] = 0$

$$= \sum_{i=1}^{|v|}(x_i - avg_v)^2 + \sum_{i=1}^{|v|}\mathbb{E}\left[(avg_v - \widehat{avg}_v)^2\right]$$

$\mathbb{V}\left(\widehat{avg}_v\right) = \mathbb{E}\left[(avg_v - \widehat{avg}_v)^2\right]$

$$= \sum_{i=1}^{|v|}(x_i - avg_v)^2 + \sum_{i=1}^{|v|}\mathbb{V}\left(\widehat{avg}_v\right)$$

Substitute $\mathbb{V}\left(\widehat{avg}_v\right)$ from equation (3.8)

$$
\begin{aligned}
&= \sum_{i=1}^{|v|}(x_i - avg_v)^2 + \sum_{i=1}^{|v|}\frac{2}{|v|\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)} \\
&= \sum_{i=1}^{|v|}(x_i - avg_v)^2 + \frac{2}{\left(\epsilon_{in}^2 + |v|\epsilon_f^2\right)} \qquad (3.9)
\end{aligned}
$$

According to equation (3.9), a bin's error can be decomposed into two terms - the generalization error, i.e. $(x_i - avg_v)^2$, and noise variance, i.e. $\frac{2}{\epsilon_{in}^2+|v|\epsilon_f^2}$. The generalization error depends on the uniformity of the bin, with higher uniformity resulting in smaller error. The noise variance depends on the bin's size, the total budget and the budget assignment. Specifically, the bigger the bin is, the smaller the variance is and less affected it is by parameters ϵ_{in} and ϵ_f. We get the worst case noise variance when ϵ_{in} and ϵ_f are equal.

Equation (3.9) allows us to calculate a bin's error, when we have access to the real data. But our goal is to create a *partitioner* that works over $\widetilde{\mathcal{H}}$. A simple solution is to compute the generalization error with the noisy versions of f_i and avg_v, i.e. $\widetilde{x}_{i,\epsilon_{in}}$ and $\widetilde{avg}_{v,\epsilon_{in}}$. But, we have to make sure that the generalization error is adjusted by a bias term such that estimator of a bin's error using $\widetilde{\mathcal{H}}$ remains unbiased. Next, we compute the bias of the generalization error, when it is estimated over $\widetilde{\mathcal{H}}$:

$$
\begin{aligned}
GenBias =& Bias\left[\sum_{i=1}^{|v|}\left(\widetilde{x}_{i,\epsilon_{in}} - \widetilde{avg}_{v,\epsilon_{in}}\right)^2\right] \\
=& \mathbb{E}\left[\sum_{i=1}^{|v|}\left(\widetilde{x}_{i,\epsilon_{in}} - \widetilde{avg}_{v,\epsilon_{in}}\right)^2 - \sum_{i=1}^{|v|}(x_i - avg_v)^2\right] \\
=& \sum_{i=1}^{|v|}\mathbb{E}\left[\left(\widetilde{x}_{i,\epsilon_{in}} - \widetilde{avg}_{v,\epsilon_{in}}\right)^2 - (x_i - avg_v)^2\right] \\
=& \sum_{i=1}^{|v|}\mathbb{E}\left[\left(x_i - avg + N_{i,\epsilon_{in}} - \sum_{j=1}^{|v|}N_{j,\epsilon_{in}}/|v|\right)^2 - (x_i - avg_v)^2\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=1}^{|v|} \mathbb{E}\left[\left(x_i - avg + N_{i,\epsilon_{in}} - \sum_{j=1}^{|v|} N_{j,\epsilon_{in}}/|v|\right)^2 - \left(x_i - avg_v\right)^2\right] \\
&= \sum_{i=1}^{|v|} \left(x_i - avg_v\right)^2 + \sum_{i=1}^{|v|} \mathbb{E}\left[\left(N_{i,\epsilon_{in}} - \sum_{j=1}^{|v|} N_{j,\epsilon_{in}}/|v|\right)^2\right] + \\
&\quad + 2\sum_{i=1}^{|v|} \left(x_i - avg_v\right)^2 \mathbb{E}\left[N_{i,\epsilon_{in}} - \sum_{j=1}^{|v|} N_{j,\epsilon_{in}}/|v|\right] - \sum_{i=1}^{|v|} \left(x_i - avg_v\right)^2
\end{aligned}
$$

Given that $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = 0$

$$
\begin{aligned}
&= \sum_{i=1}^{|v|} \mathbb{E}\left[\left(N_{i,\epsilon_{in}} - \sum_{j=1}^{|v|} N_{j,\epsilon_{in}}/|v|\right)^2\right] \\
&= \sum_{i=1}^{|v|} \left\{\mathbb{E}\left[N_{i,\epsilon_{in}}^2\right] + \frac{\sum_{j=1}^{|v|} \mathbb{E}\left[N_{j,\epsilon_{in}}^2\right]}{|v|^2} - \frac{2}{|v|}\mathbb{E}\left[N_{i,\epsilon_{in}}^2\right]\right\} + \\
&\quad - \frac{2}{|v|}\sum_{i=1}^{|v|} \mathbb{E}\left[N_{i,\epsilon_{in}}\right] \sum_{j=1, j\neq i}^{|v|} \mathbb{E}\left[N_{j,\epsilon_{in}}\right]
\end{aligned}
$$

Given that $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = 0$

$$
= \sum_{i=1}^{|v|} \left\{\mathbb{E}\left[N_{i,\epsilon_{in}}^2\right] + \frac{\sum_{j=1}^{|v|} \mathbb{E}\left[N_{j,\epsilon_{in}}^2\right]}{|v|^2} - \frac{2}{|v|}\mathbb{E}\left[N_{i,\epsilon_{in}}^2\right]\right\}
$$

Given that $\mathbb{E}\left[N_{i,\epsilon_{in}}\right] = 0$, $\mathbb{V}\left(N_{i,\epsilon_{in}}\right) = \mathbb{E}\left[N_{i,\epsilon_{in}}^2\right]$

$$
\begin{aligned}
&= \sum_{i=1}^{|v|} \left\{\mathbb{V}\left(N_{i,\epsilon_{in}}\right) + \frac{\sum_{j=1}^{|v|} \mathbb{V}\left(N_{j,\epsilon_{in}}\right)}{|v|^2} - \frac{2}{|v|}\mathbb{V}\left(N_{i,\epsilon_{in}}\right)\right\} \\
&= \sum_{i=1}^{|v|} \left\{\frac{2}{\epsilon_{in}^2} + \frac{1}{|v|}\frac{2}{\epsilon_{in}^2} - \frac{2}{|v|}\frac{2}{\epsilon_{in}^2}\right\} \\
&= \frac{2}{\epsilon_{in}^2}\sum_{i=1}^{|v|} \left\{1 - \frac{1}{|v|}\right\} \\
&= (|v| - 1)\frac{2}{\epsilon_{in}^2}
\end{aligned}
$$

Using the noisy frequencies and the bias term equation (3.9) becomes

$$UEE(v) = \sum_{i=1}^{|v|}(\widetilde{x}_{i,\epsilon_{in}} - \widetilde{avg}_{v,\epsilon_{in}})^2 - (|v|-1)\frac{2}{\epsilon_{in}^2} + \frac{2}{\epsilon_{in}^2 + |v|\epsilon_f^2}$$

The bias term encourages the creation of bigger bins. As the privacy budget ϵ_{in} decreases, the bias term increases and bigger bins are favored. In [67] (and also present in subsection 2.2.2), the authors showed how to decompose the sum of squares into the sum of the squared frequencies and the square of the sum of the frequencies.

$$UEE(v) = \sum_{i=1}^{|v|}\widetilde{x}_{i,\epsilon_{in}}^2 - \frac{\left(\sum_{i=1}^{|v|}\widetilde{x}_{i,\epsilon_{in}}\right)^2}{|v|} - (|v|-1)\frac{2}{\epsilon_{in}^2} + \frac{2}{\epsilon_{in}^2 + |v|\epsilon_f^2} \tag{3.10}$$

Thus we get the optimized formula (3.10) which can be computed in linear time, $O(|v|)$, where $|v|$ is the size of the input bin.

3.4.3 Module Implementation

In this subsection, we focus on the implementation of the modules formalized mathematically in subsections 3.4.1, 3.4.2. We shortly overview the *finalizer* based on equation (3.1). In [69] the authors presented a greedy and a dynamic programming based *partitioner*. We implemented them in a generic fashion, such that they can be used in conjunction with any bin error formula. Additionally, we extended the dynamic programming variation, in order to make it scalable for large domains.

[1] For the sake of simplicity, equations EE (3.9) and UEE (3.10) are presented to have a single parameter, i.e. v. In reality, they have three input parameters, v, ϵ_{in}, and ϵ_f.

Weighted Average Finalizer

Algorithm 4 WA Finalizer$\mathcal{V}, \mathcal{H}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p = 0, \gamma_f, \mathcal{T} = \{\})$

$\widetilde{\mathcal{H}_f} = [\ ]$ {Create empty array}
$\epsilon_{in} = \gamma_{in}\epsilon$, $\epsilon_f = \gamma_f\epsilon$
$\alpha_1 = \frac{\epsilon_{in}^2}{\epsilon_{in}^2 + |v|\epsilon_f^2}$ and $\alpha_2 = 1 - \alpha_1$
for $v \in \mathcal{V}$ **do**
  $\widehat{avg}_v = \alpha_1 \sum_{i=1}^{|v|} \widetilde{x}_{i,\epsilon_{in}} + \alpha_2 \widetilde{avg}_{v,\epsilon_f}$
  for $i \in v$ **do**
    $\widetilde{\mathcal{H}_f}[i] = \widehat{avg}_v$
  end for
end for
Return $\widetilde{\mathcal{H}_f}$

Algorithm 4, contains the logic of the WA *finalizer* component. As described in subsection 3.4.1, for each bin in $\mathcal{V}$ the *finalizer* computes the weighted average of the average based on the noisy frequencies in $\widetilde{\mathcal{H}}$ and the noisy average $\widetilde{avg}_{v,\epsilon_f}$, that is directly computed from original histogram $\mathcal{H}$ using the remainder budget ϵ_f. WA *finalizer* does not require any extra parameters, thus $\mathcal{T}$ is empty. Parameter γ_p is ignored because it assumes that the *partitioner* does not spend any budget.

Greedy Partitioner

The greedy *partitioner* is presented in pseudocode in algorithm 5. Since all the decisions are made over $\widetilde{\mathcal{H}}$, the *partitioner* spends no extra privacy budget, i.e. γ_p is zero. Recall that the input bins $\mathcal{V}$ contain a single bin for each domain value. The generalization of the *partitoner*'s logic happens through the extra parameter EF, which represents the error formula, that is used to estimate a bin's quality. From implementation perspective, we designed the error formula as a subclass of the bin class. EF's constructor has four arguments: 1) a bin or range of bins, where the range is defined over $\mathcal{V}$, 2) the noisy histogram $\widetilde{\mathcal{H}}$, 3) the privacy budget ϵ_{in} used during the initialization step, and 4) the remainder of the budget ϵ_f. At

Algorithm 5 GP($\mathcal{V}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p = 0, \gamma_f, \mathcal{T} = \{EF\}$)

```
    V' = {}, ε_in = γ_in ε, ε_f = γ_f ε
 2: v = EF(V[1], H̃, ε_in, ε_f)
    for i = 2 up to |V| do
 4:   notMergeError = v.error() + EF(V[i], H̃, ε_in, ε_f).error()
      v.add(V[i])
 6:   if v.error() > notMergeError then
        v.remove(V[i])
 8:     V' = V' ∪ v
        v = EF(V[i], H̃, ε_in, ε_f)
10:  end if
    end for
12: V' = V' ∪ v
    Return V'
```

the initialization step (line 2), the greedy algorithm creates bin v from the first element of the input bins. During every iteration, lines 3 to 11, the algorithm decides whether or not to merge v with the i^{th} element of $\mathcal{V}$, by comparing the error before and after the merge. If merging causes the error to increase, then the algorithm removes $\mathcal{V}[i]$ from v, adds v in the output bins $\mathcal{V}'$ and sets v to start from $\mathcal{V}[i]$. Adding or removing a bin from v updates v's error. The complexity of the greedy *partitioner* is $O(|\mathcal{V}|c)$, where c is the cost of the updating the error. In case of error formula (3.10), the update modifies the sum of the squared frequencies, the sum of the frequencies and the bin's size. All three operations can happen in constant time, therefore the complexity reduces to $O(|\mathcal{V}|)$.

Dynamic Programming Partitioner

Imagine a histogram $\mathcal{H}$ with domain size equal to n and an bin error formula EF, e.g. equation (3.10). Let OPT$[1, n]$ be the minimum error over all possible partitionings of $\mathcal{H}$.

OPT$[1, n]$ can be computed based on the following recursive function:

$$\text{OPT}[i,j] = \begin{cases} EF(i,j), & i = j \\ \min\left(EF(i,j), \min\limits_{k\in[i,j)} \left(\text{OPT}[i,k] + \text{OPT}[k+1,i]\right)\right), & i < j \end{cases} \tag{3.11}$$

If a bin has size equal to 1, then the minimum error is equal to the error based on EF. Otherwise, we there are two alternatives. We can create a single bin for range $[i, j]$ or we can split it into two parts and use the optimal solution for each part. Using dynamic programming, we can efficiently compute the answer to OPT$[1, n]$ by solving each sub-problem only once and saving the results in an array.

Algorithm 6 FindOptimalSolution(v, $\mathcal{S}$)

```
   S[v.start, v.end].error = v.error()
2: S[v.start, v.end].splitPoint = −1
   if v.size = 1 then
4:    Return
   end if
6: for size = 1 up to v.size − 1 do
      splitPoint = v.start + size − 1
8:    left = S[v.start, splitPoint]
      right = S[splitPoint + 1, v.end]
10:   error = left.error() + right.error()
      if error < S[v.start, v.end].error then
12:      S[v.start, v.end].error = error
         S[v.start, v.end].splitPoint = splitPoint
14:   end if
   end for
```

Algorithm 6 corresponds to a single call of function (3.11), assuming that every sub-problem has been already solved. It has two arguments: 1) a bin v, and 2) the table $\mathcal{S}$ that contains the solutions for already solved sub-problems. During the initialization step (lines 1, 2), the algorithm assumes that the best solution for range $[v.\text{start}, v.\text{end}]$ is to use a single bin, i.e. v. Therefore, it sets the error of $\mathcal{S}[v.\text{start}, v.\text{end}]$ equal to v's error and the split point equal to special value -1. If the size of the bin is equal to 1, then no further processing is required. Otherwise, it considers splitting the range into two parts. From lines 6 up to 15

the algorithm iterates over the possible sizes of the left part. If by splitting the range, the total error decreases, then the solution's split point and error are updated appropriately. The complexity algorithm 6 is $O(K)$, where K is the number of sizes that we consider for the left part.

Algorithm 7 DPP($\mathcal{V}, \widetilde{\mathcal{H}}, \epsilon, \gamma_{in}, \gamma_p = 0, \gamma_f, \mathcal{T} = \{EF\}$)

```
    S = [ , ], ε_in = γ_in ε, ε_f = γ_f ε
 2: for size = 1 up to |V| do
      v = EF(V[1 : size], ε_in, ε_f)
 4:   FindOptimalSolution(v)
      for s = 2 up to |V| − size − 1 do
 6:     e = s + size − 1
        v.remove(V[s − 1])
 8:     v.add(V[e])
        FindOptimalSolution(v)
10:   end for
    end for
12: V' = BacktrackSolution(S)
    Return V'
```

Algorithm 7 contains pseudocode for the complete dynamic programming based *partitioner*. Naturally, it has the same signature as algorithm 5. Algorithm 7 finds the best partitioning by solving the smaller problems first. At line 2, it starts from size equal to 1 and ends with size $|\mathcal{V}|$, that corresponds to finding the complete solution. During each iteration, it creates a bin v, that starts from the first element in $\mathcal{V}$ and has size equal to variable "size". In line 5, it "moves" the the bin one position to the right by removing the element at location $s-1$ and adding element at index $s + \text{size} - 1$. Since adding and removing one element to the bin has constant cost, it efficiently updates the error for the new bin. For each new bin v, it calls algorithm 6 that finds the optimal solution for the range that v occupies.

After finding the best solution for the complete domain, in line 12, it calls algorithm 8, that backtracks the best solution based on the information in $\mathcal{S}$. The backtracking process takes place with the help of the stack, that is initialized with element $\mathcal{S}[1, |\mathcal{V}|]$. While the stack is not empty, the algorithm pops the top element from the stack. If the split point in that

Algorithm 8 BacktrackSolution($\mathcal{S}$, $\mathcal{V}$)

```
    V' = {}
 2: stack = {}
    stack.push(S[1, |V|])
 4: while not stack.empty() do
      t = stack.pop()
 6:   if t.splitPoint == -1 then
        V' = V' ∪ Bin(V[t.start, t.end)
 8:   else
        stack.push(S[t.start, t.splitPoint])
10:     stack.push(S[t.splitPoint+ 1, t.end])
      end if
12: end while
    Return V'
```

element is equal to -1, then it creates a bin that contains all values from $\mathcal{V}$ between indices t.start amd t.end. On the other hand, if the split point is a positive number, then we use the best solutions for the ranges [t.start, t.splitPoint] and [t.splitPoint+ 1, t.end].

The complexity of algorithm 7 is $O(|\mathcal{V}|K^2)$. Recall that K is the number of sizes that we consider in algorithm 7 line 3 and in algorithm 6 line 6. If we consider all possible sizes, then the total complexity is $O(|\mathcal{V}|^3)$. For very large domains such complexity is impractical. We can reduce the complexity to $O(|\mathcal{V}| \log^2 |\mathcal{V}|)$ by considering only sizes that are powers of 2.

In this section, we proposed and implemented new *partitioners* and *finalizers*. In the next two sections we continue with the experimental evaluation.

3.5 Experimental Evaluation

In this section, we perform an in depth empirical evaluation of sorting based techniques. In subsection 3.5.1, we present our experimental setup based on the principles presented in DPBench [26]. Subsection 3.5.2 contains the empirical results.

3.5.1 Experimental Setup

Datasets

In [26], Hay et al. present DPBench, a framework for evaluating differential private algorithms. They identify three basic properties for every dataset $\mathbf{x}$: 1) the **domain size** n, 2) the **scale** $\|\mathbf{x}\|_1$, i.e. the sum of the counts in $\mathbf{x}$, and 3) the **shape** of the distribution $\mathbf{x}/\|\mathbf{x}\|_1$. In order to create a fair setup for comparing various datasets, they employ a **data generator**, which allows to create synthetic datasets that have the same domain size and scale, but different shapes or the same domain size and shape, but different scales.

Dataset	Domain size	Scale	# bins based on ϵ		
			1	0.5	0.1
NetTrace	1024	100K	65	37	19
Wage-Per-Hour	1024	100K	431	261	144
Amazon	500K	7.5M	311K	235K	95K
Bid-IP	500K	7.5M	266K	167K	38K
Bid-Freq	500K	7.5M	154K	58K	4K
Ipums-Inc	48K	1M	16K	11K	5K
MD-Sal-Orig	99081	136K	6K	4K	1K
MD-Sal	48K	1M	30K	19K	5K

Table 3.2: Dataset properties

Table 3.2 contains a summary of the datasets, that we used in our experiments. For each dataset, we report the domain size, the scale and the number of bins over the real data for three levels of privacy budget ϵ. The number of bins was calculated using AHP's partitioner for $\gamma = 0.5$, but any other partitioner could have been used. The number of bins offers some intuition on the shape of the underlying distribution, as it provides a sense on the uniformity of the dataset. Uniformity is what any histogramming technique attempts to leverage and, as explained in the introduction, sorting tries to create uniformity when it does not exist. If the underlying distribution is completely uniform or close to a power law distribution, e.g. NetTrace and Bid-Freq, then the number of bins is small and sorting would offer limited

benefit. On the hand, if a distribution is "random", i.e. there is big variability between the frequencies of neighbor values, then it would have little uniformity and the number of bins will be large. In that case algorithms that consider only contiguous bins would have low performance and sorting can offer the biggest benefit.

Next we go over the datasets used in our experiments. **Wage-Per-Hour** was extracted from the 1994 and 1995 population surveys conducted by the U.S. Census Bureau[1]. The original distribution is the marginal of the wage per hour. **NetTrace** [28] was derived from an IP-level network traced collected at a major university. In the original dataset, each record reports the number of external hosts connected to an internal host. The histogram is the number of records per number of external hosts. From *Wage-Per-Hour* and *NetTrace* we isolated the first 1024 values and generated a histogram with scale equal to 100K. **Amazon** dataset was collected by SNAP group [40]. It contains metadata and review information about 548,552 different products (e.g. Books, music CDs, DVDs and VHS video tapes)[2]. The distribution is the number of reviews per product. The **Bid** datasets are derived from a Kaggle competition[3] launched by Facebook, which aims to detect bots in online auctions. Both distributions are the number of bids per IP address. In case of Bid-IP the values are ordered alphabetically by the IP address, while in case Bid-Freq the values are ordered in decreasing frequency. From the Amazon and Bid datasets, we isolated the first 500,000 values and we use the data generator in order to generate synthetic datasets with scale equal to 7.5 Million. **IPUMS-Inc** is the distribution of the pre-tax income or losses from 2001 to 2011 as recorded by IPUMS [59]. **MD-Sal** datasets are based on the Maryland salary database of 2012 [4]. *MD-Sal-Orig* is the original histogram that contains a single bin per salary. From the IPUMS and MD-Sal datasets we kept the first 48,000 values and generated two datasets with 1 million records.

[1]http://archive.ics.uci.edu/ml/datasets/Census+Income
[2]http://snap.stanford.edu/data/amazon-meta.html
[3]https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot
[4]http://data.baltimoresun.com/salaries/state/cy2012/export-for-sun-users.csv

Scale-Epsilon Exchangeability

Definition 3.1 (Scale-Epsilon Exchangeability)**.** *Let* $\mathbf{p}$ *be a shape and* $\mathbf{W}$ *a workload. If* $\mathbf{x}_1 = m_1\mathbf{p}$ *and* $\mathbf{x}_2 = m_2\mathbf{p}$*, then algorithm K is scale-epsilon exchangeable if error(K(*$\mathbf{x}_1, \mathbf{W}, \epsilon_1$*)) = error(K(*$\mathbf{x}_2, \mathbf{W}, \epsilon_2$*)) whenever* $\epsilon_1 m_1 = \epsilon_2 m_2$*.*

In [26], the authors introduced the property of scale-espilon exchangeability. If an algorithm is scale-epsilon exchangeable, then it exhibits the same performance for all cases that have the same product of privacy budget and scale. Alternatively, decreasing the privacy budget ϵ by a factor α has the same effect on the performance of the algorithm as decreasing the scale by the same factor. Scale-epsilon exchangeability significantly reduces the number of experimentation configurations that we have to consider, because it creates a direct connection between the privacy budget and the scale. In addition, they prove that most of the existing histogramming techniques follow the principle of **scale-epsilon exchangeability** theoretically or empirically.

Error Metric

Definition 3.2 (Scaled Average Per Query Error)**.** *Let* $\mathbf{x}$ *be a data vector with scale* $s = \|\mathbf{x}\|_1$ *and* $\mathbf{W}$ *be a workload of q queries with real answers* $\mathbf{y}$*. Let* $\hat{\mathbf{y}} = \mathcal{K}(\mathbf{x}, \mathbf{W}, \epsilon)$ *be the noisy output of algorithm* $\mathcal{K}$*. Given an error metric L, the scaled average per query error is defined as* $\frac{1}{sq} L(\hat{\mathbf{y}}, \mathbf{y})$*.*

For our experiments we use the scaled average per query error (definition 3.2) presented in [26]. The new error measure is the average over a workload of queries normalized by the scale of the dataset. The normalization by the dataset's scale happens in order to facilitate the comparison between datasets with different scales. Intuitively, a difference of 10 between the real and the final noisy frequency is "big" , if the real frequency is equal to 10. On the

other hand, it is negligible, when the real count is close to 100,000. Unless otherwise specific, in our experiments, L_2^2 metric plays the role of error metric L.

Workloads

In DPBench, the authors use the prefix workload which consists of all range queries starting from the first value. The prefix workload offers a measure of the performance over all possible query sizes. In order to fully analyze the performance of sorting, we separate the type of workload based on the type of attribute. In case of **categorical** variables, we use the identity workload that consists of all range queries of size 1. When we evaluate **continuous** attributes, we differentiate over two types of workloads consisting of small or big range queries. The **small range** workload consists of all possible queries of size 1 to 10, while the **big range** workload consists of all possible queries with range between 100 and 1000 in increments of 100.

In the next subsection, we will use the experimental framework presented in this subsection in order to explain the advantages and disadvantages of sorting.

3.5.2 Experimental Results

Using the experimental framework presented in the previous subsection, we present the right settings to use sorting and when to avoid it. Our experiments show that the **"ideal" scenario** for sorting based techniques consists of 1) a workload composed of small range queries and 2) a dataset that has a very large domain, a large scale and a shape that is "random" and not partially or fully "sorted". Exponential or power law distributions are examples of "sorted" distributions.

All the empirical results are presented in terms of the scaled average per query error using

Algorithm	Components
Identity	Laplace mechanism with scale $1/\epsilon$
AHP	AHP's *finalizer* and bin error formula
WAF	*finalizer* and bin error formula presented in subsection 3.4.3
[G | D]X	[Greedy | Dynamic Programming] *partitioner* + algorithm X
DPC	DPCube *partitioner* + WAF *finalizer*
DAWA	Regular DAWA [12]
sDAWA	DAWA's *partitioner* + WAF *finalizer*
sX	Sorting + algorithm X
tX	Frequency thresholding + algorithm X

Table 3.3: Acronyms overview.

the L_2^2 error metric. In case of tabular representation of the results, the reader is advised to pay attention to the **error scale** present at the bottom of the table. Multiplying the value in a cell by the error scale yields the actual scaled average per query error. Before we go over the results, in table 3.3, we overview the experimental configurations and their **acronyms**. In all the experiments, the Identity mechanism, that publishes all frequencies with Laplace Mechanism, serves as a baseline. AHP and WAF refer to the *finalizer* and the bin error estimation formula used by the *partitioners*. When we prefix AHP or WAF with the letter 'G' ('D'), then the best partitioning is calculated using the greedy (dynamic programming) based *partitioner* presented in section 3.4.3. When a configuration is prefixed with letter 's', then it is enhanced with sorting. When an algorithm is preceded by letter 't', then the basic *initializer* is replaced by *initalizer* with an extra thresholding step, that sets "small" frequencies to 0, as designed in AHP. Note that cofiguration **'stAHP'** describes the complete AHP algorithm as presented in [69]. DAWA refers to the original algorithm as described in [12]. We did not implement DAWA, instead we extended the authors' code. We chose DAWA, because in [26], it is shown to have the most stable behavior when compared with other 12 techniques. The sorting enhanced version of DAWA, i.e. sDAWA, uses the DAWA's *partitioner*, but the WAF *finalizer*, because it can leverage the information learned during the initialization step. Also the privacy budget for running DAWA's *partitioner* is set to $0.25 \cdot (1 - \gamma_{in})$. DPC and sDPC configurations use DPCube's *partitioner* and

the WAF *finalizer*, because WAF is more efficient to use compared to the least square solution proposed in [62]. DPCube's *partitioner* presented an interesting alternative to our dynamic programming alternative. Additionally, by setting threshold parameter ξ to $\frac{2}{\epsilon^2}$, the *partioner*'s performance improved considerably.

The experiments in the remainder of the subsection are organized into three major groups based on the domain size and the workload type: 1) categorical attributes with large domains, 2) continuous datasets with large domains for small and large range queries and 3) small domain sizes.

Large, categorical domains

In this subsection, we examine the behavior of sorting for large categorical domains. In table 3.4, we primarily focus on how sensitive sorting is to parameter γ_{in}, the portion of the budget used by the *initializer*. In table 3.5, we vary the shape of the distribution and the privacy budget ϵ. Recall that based on scale-epsilon exchangeability, decreasing ϵ by 10 times has the same effect as reducing the scale by the same factor.

ϵ	1.0				0.1			
γ_{in}	0.3	0.5	0.7	0.9	0.3	0.5	0.7	0.9
Identity	0.27	0.27	0.27	0.27	26.67	26.67	26.67	26.67
GAHP	**0.61**	0.79	1.44	6.35	34.26	**32.56**	33.68	99.41
sGAHP	1.67	0.82	**0.79**	2.89	54.05	28.72	**23.18**	51.03
stGAHP	1.67	0.82	**0.79**	2.89	54.46	28.59	**23.18**	50.80
sDAHP	1.59	0.67	0.37	**0.24**	52.99	25.82	15.85	**10.91**
sDWAF	1.61	0.67	0.37	**0.24**	53.43	26.00	15.90	**11.26**
DPC	0.70	0.55	0.46	**0.33**	**30.59**	36.77	39.17	30.68
sDPC	1.60	0.67	0.37	**0.24**	53.31	25.92	15.88	**11.36**
DAWA	0.80	0.80	0.80	0.80	30.93	30.93	30.93	30.93
sDAWA	0.93	0.60	0.38	**0.28**	31.33	25.76	19.73	**18.03**
	error scale : 10^{-6}							

Table 3.4: Amazon dataset. Scaled average per query error as function of privacy budget ϵ and parameter γ_{in}. Cases in bold have the minimum error per row and ϵ.

The main conclusion from table 3.4 is that our proposed dynamic programming based *partitioner* reduces the error considerably over the greedy *partitioner* as shown by comparing lines sGAHP and sDAHP. Not only does sDAHP have smaller error than sGAHP, but also it wins over the baseline identity approach for $\epsilon = 1$. Note that when ϵ is equal to 1, only sorting based algorithms (excluding the greedy) win over the baseline solution. The optimal choice for parameter γ_{in} depends on the type of *partitioner*. In case of DAWA, DPCube and dynamic programming based *partitioner* the error is minimized when γ_{in} is 0.9 , while for the greedy *partitioner* is 0.7 .

Dataset	Amazon			Bid-IP			Bid-Freq		
ϵ	1.0	0.5	0.1	1.0	0.5	0.1	1.0	0.5	0.1
Identity	0.27	1.07	26.67	0.27	1.07	26.67	0.27	1.07	26.67
sDAHP	0.24	0.82	**10.91**	0.27	0.80	**6.44**	0.25	0.72	5.83
sDWAF	**0.24**	**0.82**	11.26	**0.26**	**0.78**	6.50	**0.24**	**0.70**	6.06
DPC	0.33	1.31	30.68	0.34	1.31	29.85	0.33	1.26	29.16
sDPC	0.24	0.83	11.36	0.26	0.78	6.55	0.24	0.70	6.13
DAWA	0.80	2.68	30.93	0.85	2.21	11.83	0.47	0.71	**1.51**
sDAWA	0.28	1.03	18.03	0.33	1.10	15.14	0.31	1.04	13.93
	error scale : 10^{-6}								

Table 3.5: Scaled average per query error as function of the shape. Parameter γ_{in} fixed to 0.9. Cases in bold is the minimum over the column.

In table 3.5, we examine the performance as a function of the shape and privacy budget ϵ. We only present the results for the best algorithms and parameter γ_{in} is fixed to 0.9 . In general, we observed that for very large domains γ_{in} equal to 0.9 offers the smallest error independently of the shape and ϵ. The main observation from table 3.5 is how the shape of the distribution can influence the final performance. When the distribution is "random", e.g. Amazon and Bid-IP, sorting based techniques win by far. On the other hand, when the distribution is "sorted", e.g. Bid-Freq, then sorting is not always the answer. For large values of privacy budget ϵ sorting wins, while for smaller values of ϵ a non-sorting based technique (DAWA) has the smallest error. For large domains, we did not observe a big difference between choosing AHP's or WAF's *finalizer* and bin error formula. The reason is that when the final bins are big, then the best estimator for a bins frequency is the average

learned by spending the remainder portion of the budget γ_f. In practice, WAF *finalizer* defaults to AHP's *finalizer*. Yet, we observed that WAF (AHP) is slightly better for bigger (smaller) values of ϵ.

Large, continuous domains

For the next group of experiments, we change the domain type to continuous attributes. We examine small and large range workload separately.

ϵ	1.0				0.5			
γ_{in}	0.3	0.5	0.7	0.9	0.3	0.5	0.7	0.9
Identity	0.81	0.81	0.81	0.81	3.24	3.24	3.24	3.24
GAHP	0.93	0.85	**0.43**	0.86	3.76	3.28	**1.34**	2.14
sGAHP	0.62	0.32	**0.25**	0.42	1.68	0.86	**0.61**	0.94
stGAHP	0.62	0.33	**0.24**	0.43	1.76	0.87	**0.62**	0.93
sDAHP	0.60	0.30	0.19	**0.14**	1.65	0.79	0.51	**0.38**
sDWAF	0.64	0.30	0.19	**0.18**	1.68	0.80	**0.51**	0.55
DPC	**0.69**	1.06	1.23	0.98	**2.68**	4.17	4.90	3.91
sDPC	0.61	0.31	0.19	**0.18**	1.73	0.81	**0.52**	0.59
DAWA	0.26	0.26	0.26	0.26	0.77	0.77	0.77	0.77
sDAWA	0.39	0.39	**0.36**	0.38	1.03	1.01	**1.00**	1.14
	error scale : 10^{-4}							

Table 3.6: Dataset MD-Sal-Orig. Small range workload. Scaled average per query error as function of parameter γ_{in}. Cases in bold have the minimum error per row and ϵ.

In table 3.6, we fix the dataset to MD-Sal-Orig and we observe how the behavior of the configurations, that we saw in table 3.4, change as a function of parameter γ_{in}. The performance is evaluated over the small range workload, that consists of all ranges queries with size between 1 and 10. The conclusions are similar to the ones in case of categorical attributes and the identity workload. Sorting when combined with the dynamic programming or the DPCube partitioner can greatly improve the performance for small size range queries.

In table 3.7, we fix parameter γ_{in} to 0.9 and we vary the shape of the distribution. The experimental results show that sorting helps in case of small range queries. The WA finalizer

Dataset	MD-SAL			IMPUS-INC		
ϵ	1.0	0.5	0.1	1.0	0.5	0.1
Identity	**0.11**	0.44	11.00	0.11	0.44	11.00
sDAHP	0.13	0.41	**3.63**	0.08	0.27	**2.64**
sDWAF	0.12	**0.39**	4.08	**0.06**	**0.22**	3.00
DPC	0.13	0.53	13.23	0.13	0.54	13.43
sDPC	0.12	0.40	4.40	0.07	0.22	3.32
DAWA	0.20	0.59	5.37	0.11	0.33	4.08
sDAWA	0.15	0.54	8.79	0.18	0.48	9.81
	error scale : 10^{-4}					

Table 3.7: Scaled average per query error as function of the shape for small range workloads. Parameter γ fixed to 0.9. Cases in bold are the minimum over the column.

offers an advantage for higher levels of privacy budget and AHP's finalizer for smaller levels. In case of the MD-SAL dataset and privacy budget 1.0 the Identity baseline algorithm remains the best solution. The previous results come to verify the fact that there is no algorithm that offers the best solution over all settings and meta algorithms like Pythia [39] are required in order to solve the problem of algorithm selection. Having said that, sorting based algorithms still offer performance very close to the baseline.

Dataset	MD-SAL			IMPUS-INC		
ϵ	1.0	0.5	0.1	1.0	0.5	0.1
Identity	**0.11**	0.44	10.97	0.11	0.44	10.97
sDAHP	0.14	0.59	24.87	0.45	2.37	20.46
sDWAF	0.12	0.44	15.91	0.30	1.63	14.16
DPC	0.13	0.57	14.56	0.13	0.55	13.06
sDPC	0.13	0.42	15.90	0.28	1.42	12.19
DAWA	0.11	**0.31**	**2.55**	**0.07**	**0.19**	**2.46**
sDAWA	0.30	1.11	67.98	1.03	5.27	172.15
	error scale : 10^{-2}					

Table 3.8: Scaled average per query error as function of the shape. Parameter γ_{in} fixed to 0.9. Cases in bold are the minimum over the column.

In table 3.8, we keep the experimental setting the same as with table 3.7, but we evaluate the performance in terms of large range queries. In this setting sorting does not help or it does not help in a consistent fashion. DAWA without sorting offers the best utility. We identified two reasons. Firstly, in case of large range queries, DAWA's *partitioner* is superior

ϵ	1.0				0.5			
γ_{in}	0.3	0.5	0.7	0.9	0.3	0.5	0.7	0.9
Identity	1.10	1.10	1.10	1.10	4.39	4.39	4.39	4.39
GAHP	**1.59**	2.04	2.47	6.45	**5.67**	5.95	6.05	23.16
sGAHP	2.83	1.66	**1.60**	4.66	7.19	**3.73**	3.77	7.35
stGAHP	3.99	2.68	**1.98**	3.56	10.70	5.57	**4.39**	7.33
sDAHP	2.48	1.47	**1.14**	1.85	5.93	3.25	**2.77**	4.95
sDWAF	2.66	1.46	1.10	**1.01**	7.66	3.55	**3.00**	3.71
DPC	**1.21**	1.62	1.80	1.35	**4.25**	6.64	6.80	5.44
sDPC	2.57	1.52	1.17	**1.12**	6.50	3.64	**3.03**	4.33
DAWA	1.18	1.18	1.18	1.18	3.09	3.09	3.09	3.09
sDAWA	2.54	2.16	**1.72**	1.73	5.97	5.97	**5.45**	5.58
	error scale : 10^{-4}							

Table 3.9: Dataset Wage-Per-Hour. Small range workload. Scaled average per query error as function of parameter γ_{in}. Cases in bold are the minimum error per row and ϵ.

to the frequency based *partitioners*, e.g. DPCube, AHP or WAF. Secondly, sorting works by grouping together values that in the original domain could be "very far". When a range query is small, then it is answered using a single or a small number of bins. On the other hand, a large range query will be answered by "combining" multiple bins. The bigger the number of bins used to answer a query, then larger the final error is going to be. Without sorting a big range query will be answered by only a limited number of bins.

The main conclusion from the experiments on large cateogrical domains is that sorting helps for small range queries. In case of large range queries, non-sorting based techniques like DAWA will offer better utility.

Small domains

For the last set of experiments we focus on small domain continuous datasets. In tables 3.9 and 3.10, we examine the effect of parameter γ_{in} and the shape of the distribution in case of the small range workload.

As the domain size decreases, the "difficulty" of the dataset increases. With the average bin

Dataset	WAGE-PER-HOUR			NETTRACE		
ϵ	1.0	0.5	0.1	1.0	0.5	0.1
Identity	1.10	4.39	109.84	1.10	4.39	109.84
sDAHP	1.85	4.95	**39.83**	1.69	3.95	93.39
sDWAF	**1.01**	3.71	82.05	0.89	3.36	75.26
DPC	1.35	5.44	132.36	1.43	5.07	141.11
sDPC	1.12	4.33	87.68	0.89	3.60	85.64
DAWA	1.18	**3.09**	63.45	**0.18**	**0.38**	**6.26**
sDAWA	7.35	12.96	122.89	2.66	18.66	133.68
	error scale : 10^{-4}					

Table 3.10: Scaled average per query error as function of the shape for small range workload. Parameter γ_{in} fixed to 0.9. Cases in bold are the minimum over the column.

size being small, the performance of sorting based techniques is more sensitive to the right choice parameter γ_{in} as a function of privacy budget ϵ. The smaller ϵ is, the best performance is achieved for a smaller value of γ_{in}. Additionally the gap between WAF and AHP *finalizer* increases, with WAF being considerably better (worse) than AHP's *finalizer* when the privacy budget is big (small). Table 3.10 re-affirms that when the shape of the distribution is "sorted" then DAWA has better performance. On the hand, the more "random" a distribution is the the more beneficial it is to choose a sorting based solution.

Chapter 4

One-sided Privacy

4.1 Introduction

Public and private organizations capture and store diverse types of information about individuals that they are obligated to publish, or they could benefit from sharing with others [31]. A key impediment in sharing is the concern of privacy. For instance, information about visited locations, online purchases, movie preferences [58], or web searches [7] has been shown to lead to inferences about sensitive properties like health, religious affiliation, sexual orientation, or family situation.

In recent years, the field of privacy preserving data-sharing has flourished and *differential privacy* [21] (DP) has emerged as the predominant privacy standard. Informally, DP states that for a database that contains a single record per user, the participation of an individual will change the output of an analysis by a bounded factor proportional to a privacy parameter ϵ. Most work in differential privacy (with the exceptions of [6, 32]) has been developed under the assumption that every record in the database is *equally* sensitive.

In this work, we depart from the above well trodden path, as we consider data sharing when only part of the data is sensitive, while the remainder (that may consist the majority) is non-sensitive. In real-world scenarios, the sensitive/non-sensitive dichotomy of data is not always apparent, but in certain cases it naturally occurs as a consequence of legislature, personal preferences, convenience, and privacy *policies*.

Example 4.1. *The General Data Protection Regulation [52, 46] (GDPR), that will take effect in European Union (EU) in 2018, imposes strict rules on how information about EU citizens is handled by companies. For instance, an individual must provide an affirmative consent (opt-in) in order to allow the collection and analysis of his/her data. Additionally, information about minors under 16 years of age can not be stored and processed without parental authorization. Failure to comply with GDPR could ensue a fine of up to 4% of the company's annual turnover. The previous legislative rules can easily be used to classify a citizen's record as sensitive or not.*

Example 4.2. *In psychology literature, it is well established that privacy attitudes depend on personal preferences and convenience [2, 3]. In a large scale study in e-commerce [9], researchers found two major groups of individuals: those who had little to no privacy concerns and the "privacy fundamentalists". Moreover, individuals' tendency to opt-in or opt-out of data sharing is a function of perception of utility offered by an application versus the cost of privacy leakage. Thus, if the gain is sufficient, a big portion of users will willingly release their information.*

Example 4.3. *We examine the use case of privacy applications in a smart building management system [17] (SBMS). Such systems, are capable of tracking individuals using indoor localization sensors and devices. The collected information is used to improve the performance of HVAC (heating, ventilation, and air conditioning) subsystems or offer location based services to the tenants, such as allowing to locate each other. SBMS need to comply with legislature, e.g. GDPR, and privacy policies. For instance, individuals should not be*

tracked in certain locations (e.g. restrooms, smoker lounge, etc.). Additionally, "privacy fundamentalists" can opt-out from the tracking system, at the expense of enjoying only limited services.

In the context of this chapter, we use policies as the language for specifying sensitivity. Traditionally, query answering in the presence of policies has been the focus of access control literature [11, 56, 57]. The *Truman* and *non-Truman* models [56] are the two primary models for query answering in access control. Based on Truman model, queries are transparently rewritten to ensure users only see what is contained in the authorized view. Thus, the result is correct only with respect to the restricted view. On the other hand, in a non-Truman model, queries that can not be answered correctly are rejected. However, neither approach guarantees privacy. Consider the SBMS example, wherein the smoker lounge is the only sensitive location and Alice is attempting to find Bob. If Bob is at the sensitive location, the Truman model will return nothing and the non-Truman model will honestly reject the query. Therefore, in both cases, Bob's location is indirectly revealed, because the sensitivity of his record is correlated with the value (in this case location) of the record. This attack is an example of what we call an *exclusion attack*.

An alternative solution is to directly apply differential privacy. Such an approach would hide all properties of a record (including its sensitivity) and protect against any exclusion attacks. However, DP algorithms will not leverage on the fact that part of the data is non-sensitive to lower the error rates of the query answers. In prior work, heterogeneous DP [6] and personalized differential privacy (PDP) [32] have tried to take advantage of different privacy levels between records. Specifically, PDP [32] considers the case where each record r is associated with its own privacy parameter $\epsilon(r)$. PDP, for instance, allows suppressing records with strong privacy requirements ($\epsilon(r) < t$), and analyzing only the records with weaker privacy requirements using a t-DP algorithm. In this context, if we model non-sensitive records by setting their privacy level to ∞, PDP would release all the non-sensitive

records unperturbed (which is an ∞-DP algorithm), which results in an exclusion attack.

In general, the fundamental problem ignored by prior work is that an individual's privacy attitude (and hence the fact that their records is sensitive or non-sensitive) can be correlated with attributes in their record (and whether they are present or absent in the database). For instance in [50], the authors note that an individual's privacy preferences are correlated to demographic attributes like income, age, and political views.

In this chapter, we make the following contributions:

1. Motivated by the shortcomings of existing solutions, we introduce *one-sided privacy* (OSP), a novel privacy definition that provides rigorous privacy guarantees on the sensitive portion of the data, and guards against exclusion attacks.

2. We develop a variant of the randomized response algorithm, that can truthfully release a sample of the non-sensitive data, while satisfying OSP. We use this to release and analyze mobility trajectories of users in the context of smart buildings. DP algorithms are unable to analyze these data with low error. In contrast, data released under OSP are able to support these analyses with reasonably high accuracy, especially when the fraction of sensitive data is small.

3. We propose a new primitive for answering count queries under OSP.

The remainder of the chapter is organized as follows. We define one-sided privacy in Section 4.2, prove its composition properties, and discuss its relation to similar privacy definitions. In Section 4.2, we also formalize the exclusion attack and we show how OSP protects against it. Section 4.3 contains an algorithm for releasing true data under OSP. In Section 4.4, we shift our focus on releasing counting queries under OSP. In Section 4.5, we empirically evaluate our algorithms.

4.2 One-Sided Privacy

In this section, we formally define one-sided privacy (OSP). In subsection 4.2.2, we define the exclusion attack, and we prove that OSP protects against it. Furthermore, in subsection 4.2.3, we examine the composition properties of OSP and in 4.2.4 we compare and contrast OSP with related work.

4.2.1 Definition

Definition 4.1 (Policy Function)**.** *Let $r \in \mathcal{T}$ be a record. A policy function $P : \mathcal{T} \to \{0, 1\}$ denotes a record $r \in \mathcal{T}$ is either sensitive ($P(r) = 0$) or that it is non-sensitive ($P(r) = 1$).*

A *policy function* (or simply policy) classifies database records as either sensitive or non-sensitive, examples of policy functions include:

- $\lambda r.if(r.Age \leq 17) : 0; else : 1$ encodes the policy that records corresponding to minors are sensitive.
- $\lambda r.if(r.Race = NativeAmerican \vee r.Optin = False) : 0; else : 1$ implies that all records who have either opted out, or are native Americans are sensitive.

Policies considered in this work are *non-trivial* i.e., they result in at least one sensitive and one non-sensitive record. If all records are sensitive the private analysis can be done with standard DP algorithms. Conversely, if all records are non-sensitive then any non-private algorithm can be used.

As stated earlier, our goal is to design a privacy definition that provides a rigorous privacy guarantee for the sensitive records (i.e., $\{r : P(r) = 0\}$) alone. Towards that goal, we now define one-sided neighboring databases under policy P.

Definition 4.2 (One-sided P-Neighbors)**.** *Let D, D' be two databases and P a policy function with $r \in D$ a sensitive record, i.e. $P(r) = 0$, and $r' \in D' \backslash \{r\}$. Database D' is a one-sided P-neighbor of D, i.e. $D' \in N_P(D)$, if and only if $D' = D \backslash \{r\} \cup \{r'\}$.*

One-sided neighbors under policy P are created by replacing a single sensitive record with any other possible record. This implies that the N_P relation is asymmetric, i.e., $D' \in N_P(D)$ does not imply $D \in N_P(D')$. For instance, if D contains only non-sensitive records then $N_P(D) = \emptyset$. On the other hand, D is the P-neighbor of every database D' with the same size as D, that differs from D by one sensitive record.

Definition 4.3 (One-sided Privacy)**.** *Let $\mathcal{M}$ be a randomized algorithm. Algorithm $\mathcal{M}$ satisfies (P, ϵ)-one-sided privacy, if and only if $\forall \mathcal{O} \subseteq range(\mathcal{M})$, and $\forall D, D' \in N_P(D)$:*

$$\Pr[\mathcal{M}(D) \in \mathcal{O}] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in \mathcal{O}] \tag{4.1}$$

Where the probabilities are taken over coin tosses of $\mathcal{M}$.

A mechanism satisfies OSP, if the likelihood of any output does not change by too much when a sensitive record is replaced with any other record. Essentially, OSP provides an indistinguishability property similar to DP, but only for the sensitive records. An OSP mechanism not only "hides" the value of any sensitive record, but also the fact that it is sensitive or not. As we will see in the following subsection, this key result allows OSP to protect against exclusion attacks. Lastly, OSP does not constraint the information leakage of non-sensitive records as long as the privacy of the sensitive records is preserved.

4.2.2 OSP and the Exclusion Attack

In this subsection, we (a) formalize the notion of freedom from exclusion attacks, (b) show that prior work fails to satisfy it, and (c) show that all one-sided privacy algorithms satisfy

it.

Exclusion Attacks: As discussed earlier, a common approach to share data with records of varying sensitivity levels is to suppress all sensitive records and release all the non-sensitive record unperturbed. This approach can lead to privacy leaks on sensitive records. Consider the introductory example on SBMS, wherein the smoker lounge is the only sensitive location. Alice knows that Bob is in the database, but she is not aware of Bob's location. If Bob's location is not released, then she immediately learns that he is at the smoker's lounge based on the policy. We now formalize *freedom from exclusion attack*.

Definition 4.4 (Freedom from Exclusion Attacks)**.** *A mechanism $\mathcal{M}$ satisfies freedom from exclusion attacks for policy P with privacy level ϵ if:*

$$\forall x : P(x) = 0, \forall y \in \mathcal{T}, \forall \mathcal{O} \subseteq range(\mathcal{M})$$

$$\forall \theta : \Pr{}_\theta(r = x) > 0, \Pr{}_\theta(r = y) > 0$$
$$\frac{\Pr{}_\theta(r = x | \mathcal{M}(D) \in \mathcal{O})}{\Pr{}_\theta(r = y | \mathcal{M}(D) \in \mathcal{O})} \leq e^\epsilon \frac{\Pr{}_\theta(r = x)}{\Pr{}_\theta(r = y)}$$

Where r is the target record, x is a value that makes the record sensitive, y is another value in the domain, and θ is the adversary's prior about the database.

Informally, a mechanism $\mathcal{M}$ satisfies freedom from exclusion attacks if an attacker's certainty about whether a target record is sensitive increases by at most a bounded amount after accessing $\mathcal{M}$'s output. More specifically, the definition states that the adversary should not be able to significantly reduce his uncertainty about whether a record is sensitive after seeing the output of the mechanism. It follows, that a mechanism that reveals all non-sensitive records is not free from exclusion attacks, since there are datasets and outputs where $P_\theta(r = y | \mathcal{M}(D) \in O)$ becomes 0 for values y that are non-sensitive according to the

policy, resulting in an unbouned posterior odds-ratio.

OSP and Freedom from Exclusion Attacks: Any mechanism that satisfies (P, ϵ)-OSP also satisfies freedom from exclusion attacks as long as the attacker does not know any correlations between the target record and the rest of the database.

Theorem 4.1. *A mechanism satisfying (P, ϵ)-OSP satisfies freedom from exclusion attacks for policy P and privacy level ϵ, for all adversaries whose prior knowledge can be expressed as a product of priors over individual records.*

Proof. Since the adversary's prior can be decomposed into a product of priors over individual records, we can show that:

$$\frac{\Pr_\theta(r = x|\mathcal{M}(D) \in \mathcal{O})}{\Pr_\theta(r = y|\mathcal{M}(D) \in \mathcal{O})} \leq \frac{\Pr_\theta(r = x)}{\Pr_\theta(r = y)} \max_{D,D'} \frac{\Pr(\mathcal{M}(D) \in \mathcal{O})}{\Pr(\mathcal{M}(D') \in \mathcal{O})}$$

where D is a database where record r takes the value x and D' is a database where record r takes the value y. There are precisely neighboring databases under OSP (when x is a sensitive record, and y is an arbitrary record). Thus,

$$\frac{\Pr_\theta(r = x|\mathcal{M}(D) \in \mathcal{O})}{\Pr_\theta(r = y|\mathcal{M}(D) \in \mathcal{O})} \leq \frac{\Pr_\theta(r = x)}{\Pr_\theta(r = y)} \cdot e^\epsilon$$

This completes the proof. □

The above proof also shows that all DP mechanisms also satisfy freedom from exclusion attacks for any policy.

The assumption that the adversary does not know any correlations between the target record and the rest of the database is critical. Prior work has shown a "No Free Lunch Theorem"

[37] that states that any mechanism that ensures properties of records are protected against all possible adversaries can not necessarily provide any useful information about the data. To illustrate this, consider an ϵ-differentially private mechanism that releases the number of non-sensitive records (using the Laplace mechanism). Suppose the adversary knows that either all the records are sensitive, or all the records are non-sensitive. Under such a strongly correlated prior, the adversary can tell with high probability whether a target record is sensitive (when the noisy count is close to 0) or not. Thus, useful mechanisms can only ensure freedom from exclusion attacks under some assumption on the attacker prior. We prove freedom from exclusion attack under the independence assumption (which is the standard assumption used when proving semantic privacy properties of differential privacy [38]). Considering freedom from exclusion attacks when records are correlated is an interesting direction for future work.

4.2.3 Composition

Much like DP, one-sided privacy also composes with itself. First we define policy relaxations, which will help us to compose OSP instantiations with different policies.

Definition 4.5 (Policy Relaxation)**.** *Let P_1, P_2 be two policy functions. P_1 is a relaxation of P_2, denoted as $P_1 \succeq_p P_2$, if and only if for every record r, $P_1(r) \geq P_2(r)$.*

If P_1 is a relaxation of P_2, then every record that is sensitive under P_1 is also sensitive under P_2. We also say that P_1 is *weaker* than P_2 or P_2 is *stricter* than P_1. Relaxing the policy also results in a weaker privacy definition.

Theorem 4.2 (Privacy Relaxation)**.** *Let $\mathcal{M}$ be a (P_2, ϵ)-one-sided private mechanisms, . If $P_1 \succeq_p P_2$, then $\mathcal{M}_2$ also satisfies (P_1, ϵ)-one-sided privacy.*

Proof. If $P_1 \succeq_p P_2$, then for every database D, $N_{P_1}(D) \subseteq N_{P_2}(D)$. Thus based on Definition

4.2, $\mathcal{M}_2$ also satisfies (P_1, ϵ)-one-sided privacy. □

We next define the minimum relaxation of a pair of policies that are not relaxations of each other.

Definition 4.6 (Minimum Relaxation)**.** *Let $P_1, \ldots, P_k$ be a set of policy functions. We define the minimum relaxation as the policy function P_{mr} such that for every record r, $P_{mr}(r) = \max(P_1(r), \ldots, P_k(r))$.*

The minimum relaxation of two policies, P_1, P_2 classifies as sensitive the records that are sensitive based on both policies. Records that are deemed non-sensitive by either P_1 or P_2 are considered non-sensitive by $P_{mr}(r)$. P_{mr} is the strictest policy that is a relaxation of both P_1 and P_2. When $P_1 = P_2$, then $P_{mr} = P_1 = P_2$. We are now ready to define the sequential and parallel composition theorems for OSP.

Theorem 4.3 (Sequential Composition)**.** *Let $\mathcal{M}_1, \ldots, \mathcal{M}_k$ be $(P_1, \epsilon_1), \ldots, (P_k, \epsilon_k)$-one-sided private algorithms and P_{mr} the minimum relaxation of $P_1, \ldots, P_k$. Then the composition of mechanisms $\mathcal{M}_C = \mathcal{M}_1 \circ \ldots \circ \mathcal{M}_k$ satisfies (P_{mr}, $\sum_i \epsilon_i$)-one-sided privacy.*

Proof. According to Theorem 4.2, every algorithm $\mathcal{M}_i$ satisfies (P_{mr}, ϵ_i)-one-sided privacy. Let D be a database and $D' \in N_{P_{mr}}(D)$, then

$$\frac{\Pr[\mathcal{M}(D) = (x, y)]}{\Pr[\mathcal{M}(D') = (x, y)]} = \prod_i \frac{\Pr[\mathcal{M}_1(D) = x]}{\Pr[\mathcal{M}_1(D') = x]} \leq e^{\sum_i \epsilon_i}$$

Thus $\mathcal{M}$ satisfies (P_{mr}, $\sum_i \epsilon_i$)-one-sided privacy. □

Theorem 4.4 (Parallel Composition)**.** *Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a disjoint partitioning and $\{D_1, \ldots, D_n\}$ be its instantiation over database D. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be one-sided private mechanisms, with $\mathcal{M}_i$ running on S_i and satisfying (P_i, ϵ_i)-one-sided privacy. Let P_{mr} be the*

minimum relaxation of P_1, ..., P_n. Then the composition of $\mathcal{M}_1 \circ \mathcal{M}_2 \circ \ldots \circ \mathcal{M}_k$ satisfies $(P_{mr}, \max_{i,j \neq i}(\epsilon_i + \epsilon_j))$-one-sided privacy.

Proof. Based on Theorem 4.2 every mechanism $\mathcal{M}_i$ satisfies (P_{mr}, ϵ_i)-one-sided privacy. Let D and $D' \in N_{P_{mr}}(D)$ be two neighboring databases that differ on records r, r'. Also, let $\{D_1, \ldots, D_n\}$ and $\{D'_1, \ldots, D'_n\}$ be the instantiation of partitioning $\mathcal{S}$ for D and D' respectively. In the worst case r, r' belong to different partitions, because they have different values or sensitivity. Without loss of generality let r belong to D_1 and r' be part of D'_2. For any index bigger that 2, D_i is the same as D'_i. Therefore we have:

$$\frac{\Pr[\mathcal{M}(D) = (x_1, \ldots, x_n)]}{\Pr[\mathcal{M}(D') = (x_1, \ldots, x_n)]} =$$
$$\frac{\Pr[\mathcal{M}_1(D_1) = x_1] \times \ldots \times \Pr[\mathcal{M}_n(D_n) = x_n]}{\Pr[\mathcal{M}_1(D'_1) = x_1] \times \ldots \times \Pr[\mathcal{M}_n(D'_n) = x_n]} =$$
$$\frac{\Pr[\mathcal{M}_1(D_1) = x_1] \Pr[\mathcal{M}_2(D_2) = x_2]}{\Pr[\mathcal{M}_1(D'_1) = x_1] \Pr[\mathcal{M}_2(D'_2) = x_2]} \leq e^{\epsilon_1 + \epsilon_2} \leq e^{\max_{i,j \neq i}(\epsilon_i + \epsilon_j)}$$

Therefore $\mathcal{M}$ satisfies $(P_{mr}, \max_{i,j \neq i}(\epsilon_i + \epsilon_j))$-one-sided privacy. □

4.2.4 Connection with other Privacy Definitions

In this subsection, we discuss the connection of OSP with other three similar privacy definitions: differential privacy, personalized differential privacy and blowfish privacy.

Differential Privacy: OSP is a generalization of differential privacy; the latter is a special case of the former when every record is sensitive as shown below.

Definition 4.7 (All Sensitive Policy)**.** *Let P_{all} denote the policy function that classifies every record as sensitive.That is, $P_{all}(r) = 0$, $\forall r$. We call this the* all sensitive policy*.*

Corollary 4.2.1. *Let $\mathcal{M}$ be an ϵ-differential private mechanism. Then, $\mathcal{M}$ satisfies also (P, ϵ)-one-sided privacy for every policy function P.*

Proof. An ϵ-differentially private algorithm satisfies (P_{all}, ϵ)-one-sided privacy. Every policy P is a relaxation of P_{all}. Thus, by Theorem 4.2, a mechanism that satisfies ϵ-differential private also satisfies (P, ϵ)-one-sided privacy. □

Differential privacy is the strictest form of one-sided privacy and any differential private algorithm will also satisfy any version of one-sided privacy for the same ϵ.

Personalized Differential Privacy (PDP) [32]: In PDP different records in the dataset are allowed different levels of privacy. This is modeled by a publicly known function Φ that associates records r with a personalized privacy level $\Phi(r) = \epsilon_r$. Moreover, PDP requires not only that Φ is public but also that "*a record's privacy parameter must not indicate anything about their sensitive values*"[32], i.e., the values of Φ for each record in the dataset. In contrast, OSP does not disclose the privacy parameter of its records. As discussed in Section 4.1, OSP is motivated by scenarios where a user's privacy preference is correlated to their data values. The above distinction further explains the fact that PDP algorithms are susceptible to exclusion attacks.

The PDP preserving *Threshold* algorithm [32] works as follows: first it picks a threshold t and suppresses all records with privacy parameter $\epsilon < t$, then it runs a t-differentially private computation on the rest of the records. We could model OSP using PDP, by associating sensitive records with a bounded privacy parameter ϵ, and non-sensitive records with a privacy parameter of ∞. In this case, the Threshold algorithm will suppress all sensitive records, while releasing the non-sensitive records with no perturbation. It naturally follows from Definition 4.4 that this algorithm is vulnerable to exclusion attacks.

Blowfish Privacy [23, 30]: Recent work has considered generalizations of differential privacy by modifying the notion of neighboring datasets. One example is Blowfish Privacy, where the notion of neighboring datasets is captured using a graph theoretic construct also called a *policy*. Unlike an OSP policy, which encodes whether a record is sensitive or not,

a Blowfish privacy policy is used to specify which properties of records are sensitive (for instance, whether neighboring databases must differ in the entire record, or just in a single attribute of a record). While OSP policies provide varied privacy protection to different records, Blowfish policies permit the same level of privacy for all records. Finally, unlike OSP, in Blowfish privacy the resulting neighboring database relation is *symmetric* (like under DP).

4.3 Releasing True Data

In this section, we introduce OspRR, a randomized response based algorithm that preserves one-sided privacy. OspRR works by releasing a true sample of the non-sensitive records in the database. The ability to output a true data sample, while ensuring privacy, enables a new class of privacy preserving analysis tasks that require true data records. Examples of such tasks include the creation of *extractive summaries* [44, 65] that return a "representative" sample of the input data. Extractive summaries are common in applications such as image summarization and sharing, as well as, summarizing reviews in services such as Yelp and Amazon. Other tasks, that can greatly benefit from a sample of true data, include complex analyses, e.g. classification, or releasing histograms over very large domains and with very large sensitivity. In Section 4.5.3, we will empirically show the advantages of using OspRR for the last two cases.

Algorithm 9 OspRR (D, P, ϵ)

```
1: S ← {}
2: for r ∈ D do
3:    if P(r) = 1 then
4:       S ← S ∪ {r} with probability 1 − e^{−ε}
5:    end if
6: end for
7: return S
```

OspRR is described in Algorithm 9. The inputs to OspRR are a database D, a policy function P, and the privacy parameter ϵ. In lines 2 to 6, OspRR iterates through every record in the database. Non-sensitive records, i.e. $P(r) = 1$, are added to the output set S with probability equal to $1 - e^{-\epsilon}$. In all other cases the record is ignored.

Theorem 4.5. OspRR *satisfies* (P, ϵ)*-OSP.*

Proof. Let D be a database with size equal to 1, i.e., it contains a single record. In that case we represent neighboring databases as records r, r'. There are only two possible outputs, either release record r or suppress it (that we denote as $\text{OspRR}(r) = \varnothing$). If r is non-sensitive then r has no neighbors (see Definition 4.2). Thus, we are interested only in cases that r is sensitive ($P(r) = 0$) and r' is any possible record, such that $r \neq r'$. Based on OspRR's possible outputs, we have the following cases:

Case 1: $\text{OspRR}(r) = r$ *and* r *is sensitive.* If r is sensitive $\Pr[\text{OspRR}(r) = r] = 0$, thus equation (4.1) holds trivially.

Case 2.1: $\text{OspRR}(r) = \varnothing$ *and* r, r' *are sensitive.*

$$\frac{\Pr[\text{OspRR}(r) = \varnothing]}{\Pr[\text{OspRR}(r') = \varnothing]} = 1 \leq e^{\epsilon}$$

Case 2.2: $\text{OspRR}(r) = \varnothing$ *and* r *is sensitive,* r' *is non sensitive.*

$$\frac{\Pr[\text{OspRR}(r) = \varnothing]}{\Pr[\text{OspRR}(r') = \varnothing]} = \frac{1}{e^{-\epsilon}} = e^{\epsilon}$$

This concludes the proof for the case that $|D| = 1$. Next, we extend the proof for any possible database size. Let D, D' be two neighbors, that differ on the k^{th} record. Let $S = \{s_1, s_2, \ldots\}$ be a possible output, with s_i being the output for the i^{th} record. Since OspRR makes independent decisions for each record, we have the same result whether

Table 4.1: Percentage of released non-sensitive (ns) records using OspRR vs ϵ.

privacy parameter ϵ	1.0	0.5	0.1
% of released ns records	$\sim 63\%$	$\sim 39\%$	$\sim 9.5\%$

OspRR is applied to the complete database or each record separately. Thus we have

$$\frac{\Pr[\text{OspRR}(D) = S]}{\Pr[\text{OspRR}(D') = S]} = \frac{\Pr[\text{OspRR}(r_k) = s_k]}{\Pr[\text{OspRR}(r'_k) = s_k]} \leq e^{\epsilon}$$

This concludes the proof. □

The probability distribution of the sample size, released by OspRR, is the Bernoulli distribution with probability of success equal to $1 - e^{-\epsilon}$ and number of trials equal to the number of non-sensitive records. Table 4.1 shows the expected percentage of released non-sensitive data as a function of privacy parameter ϵ. For ϵ equal to 1.0, the percentage is as high as 63%, while for low epsilon values ($\epsilon = 0.1$), the percentage drops to approximately 9.5%.

4.4 Answering Counting Queries

We shift our focus on the task of releasing histograms over a dataset. We define a histogram query to be a set of counts defined over a non-overlapping partitioning of the dataset. One could think of a histogram query as:

SELECT group, COUNT(*)

FROM TABLE WHERE ⟨condition⟩

GROUP BY ⟨keys⟩

where the output reports all groups with both zero and non-zero counts.

Histogram estimation under differential privacy is a well researched area with rich literature to draw from. Hay et al [27] present a near-complete benchmark comparison of differentially private algorithms. The algorithms examined in [27] are sophisticated, over-engineered algorithms that permit the release of accurate low-dimensional histograms (specifically, 1- or 2-dimensional histograms with the number of bins in the thousands). As discussed earlier, any DP algorithm for histogram release will satisfy OSP. In this section, we develop OSP algorithms that can leverage the presence of non-sensitive records to provide lower error than even the state-of-the-art DP algorithms for this task.

One would expect that in the presence of non-sensitive records histogram release becomes automatically easier, but this is not the case as we identify two main issues with releasing histogram estimates under OSP. First, even in the presence of a *single* sensitive record the global sensitivity of releasing a count is still 1 (and the sensitivity of a histogram is still 2). In the remainder of this section, we will show that the sensitivity can be reduced by answering queries only on non-sensitive records. This helps when some of the bins in the histogram have exclusively sensitive records and some bins have exclusively non-sensitive records. We can use a DP algorithm for the bins with sensitive records, and an OSP algorithm (from Section 4.4) for bins with non-sensitive records.

OspRR can be used to release histograms by running the query on the sample of non-sensitive records output by OspRR. However, this approach can be worse than the Laplace mechanism in many cases.

Theorem 4.6. *Let $n = |D|$ denote the number of records in database D, and d denote the number of bins in the histogram query. Then the expected L_1 error of computing the histogram on the output of an* OspRR *algorithm that satisfies (P, ϵ)-OSP is higher than the*

expected L_1 error of Laplace mechanism that satisfied ϵ-DP, whenever:

$$n \cdot \epsilon > 2d \cdot e^{\epsilon} \tag{4.2}$$

Proof. The expected L_1 error of Laplace mechanism is $2d/\epsilon$. In the case of OSP, error is due to (a) the suppression of all sensitive records, and (b) suppression of $n \cdot e^{-\epsilon}$ non-sensitive records. Even if the number of sensitive records tends to zero, the L_1 error of OspRR is at least $n \cdot e^{-\epsilon}$. □

For instance, a 2-d histogram query over a uniform grid that divides each dimension into 100 disjoint bins would have $d = 10^4$. When $\epsilon = .1$, OspRR will have higher error than the Laplace mechanism if $n > 2.2 \times 10^5$.

For the remainder we use $\mathbf{x}$ to denote the histogram over all records in D, and $\mathbf{x}_{ns}$ over the non-sensitive records of D alone. Note that under OSP, a database $D' \in N_P(D)$ is a neighbor of D in one of two ways: (a) a sensitive record in D is replaced with another sensitive record in D', or (b) a sensitive record in D is replaced with a non-sensitive record in D'. Thus, if $\mathbf{x}_{ns}$ ($\mathbf{x'}_{ns}$) denotes the output of a histogram query over the *non-sensitive records* of D (D', resp.), then the neighboring histograms differ in at most one count, and all counts in $\mathbf{x'}_{ns}$ are no smaller than the counts in $\mathbf{x}_{ns}$. Thus, we can release histogram counts on the non-sensitive records by adding one-sided Laplace noise to ensure OSP. The one-sided Laplace distribution, $Lap_{-}(\lambda)$, is the mirrored version of exponential distribution with scale equal to λ, where all the mass is on the negative values.

Definition 4.8 (One-Sided Laplace Distribution)**.** *The probability density function of the One-Sided Laplace Distribution is:*

$$f_{Lap_{-}}(x;\lambda) = \begin{cases} \frac{1}{\lambda}\exp\left(\frac{x}{\lambda}\right) & \text{if } x \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

The OspLaplace answers queries by adding one-sided noise to the query answer computed only using non-sensitive records.

Definition 4.9 (OspLaplace)**.** *Let D be a database, P a policy, $D_{ns} = \{r \in D \mid P(r) = 1\}$ the subset of non-sensitive records. The* OspLaplace *answers a histogram query with d bins, by computing the histogram on the non-sensitive records D_{ns}, and adding a vector of d i.i.d. random variables drawn from $Lap_{-}(\lambda)$.*

Theorem 4.7. *For $\lambda = 1/\epsilon$* OspLaplace *releases histograms under (P, ϵ)-OSP.*

Proof. Let D, D' P-neighboring databases. Let $\mathbf{x}$ ($\mathbf{x}'$) denote the real **non-sensitive** histogram on D (D'). For simplicity, let $\mathcal{M}$ be an alias for OspLaplace. Note that $range(\mathcal{M}(D, P))$ $\subseteq range(\mathcal{M}(D', P))$ since $\forall i$, $x_i \leq x'_i$. Let $\mathbf{y} \in range(\mathcal{M}(D', P))$, then there are two cases:

Case 1. $\exists i : x_i < y_i \leq x'_i$. Then, $\Pr[\mathcal{M}(D) = \mathbf{y}] = 0$, thus equation (4.1) holds trivially.

Case 2. $\forall i : y_i \leq x_i$

$$\frac{\Pr[\mathcal{M}(D) = \mathbf{y}]}{\Pr[\mathcal{M}(D') = \mathbf{y}]} = \frac{\prod_i \exp\left(\epsilon \cdot (y_i - x_i)\right)}{\prod_i \exp\left(\epsilon \cdot (y_i - x'_i)\right)} \leq \exp\left(\epsilon \cdot \|\mathbf{x}' - \mathbf{x}\|_1\right) \leq \exp(\epsilon)$$

□

It is easy to see that the OspLaplace mechanism introduces noise with 1/8 the variance of the differentially private Laplace mechanism. The exponential distribution has half the variance, and another factor of 4 reduction in variance comes from the sensitivity dropping from 2 (in the case of DP) to 1 (in the case of OSP). The error can be further reduced by using the following algorithm that we call OspLaplaceL1 that leverages the fact that all input counts are non-negative.

Step 2 of the algorithm sets all negative counts after adding noise to 0. Note that after this step, every count in $\mathbf{x}_{ns}$ that was originally 0 is also output as 0. However, there

Algorithm 10 OspLaplaceL1 $(\mathbf{x}_{ns}, \epsilon)$

1: $\widetilde{\mathbf{x}}_{ns} = \mathbf{x}_{ns} + Lap_{-}(1/\epsilon)^d$
2: $\widetilde{\mathbf{x}}_{ns}[\widetilde{\mathbf{x}}_{ns} < 0.0] = 0.0$
3: $median = -\ln(2) \cdot 1/epsilon$
4: $\widetilde{\mathbf{x}}_{ns}[\widetilde{\mathbf{x}}_{ns} > 0.0] -= median$
5: **return** $\widetilde{\mathbf{x}}_{ns}$

could be other bins that are output as 0 (since their noisy count was negative). Next, note that the one-sided Laplace distribution is not unbiased, step 4 adds back the median value of a one-sided Laplace random variable to make the positive noisy counts unbiased. While OspLaplace and OspLaplaceL1 add significantly lower noise than the Laplace mechanism, they can have higher overall error than the Laplace mechanism since they only use the non-sensitive records to answer queries.

4.5 Experiments

We use a real dataset to empirically demonstrate that the OSP algorithms presented in this paper permit accurate data analysis of private datasets by leveraging the presence of non-sensitive records. We show:

- OspRR allows the release of true data records, which permits building classifiers with high accuracy *close to that of a non-private baseline* (Section 4.5.3). In contrast, a state of the art DP classifier (that considers all the data as sensitive) has accuracy close to that of a baseline that does not have access to the data.
- OspRR's output also permits releasing high dimensional histograms with *an order of magnitude smaller error* than that of a DP algorithm on these data (Section 4.5.3)
- We also consider answering low dimensional histogram queries, a task where over-engineered DP algorithms achieve highly competitive error rates. We show that our

proposed OSP algorithms outperform state of the art DP algorithms (Section 4.5.3).

We describe our dataset and policies in Section 4.5.1, the analysis tasks in Section 4.5.2, and the results in Section 4.5.3.

4.5.1 Dataset and Privacy Policies

We use TIPPERS, a real data set from a live IoT testbed under development at UCI [1].

Dataset: The TIPPERS dataset consists of a trace of distinct devices connected to 64 Wi-Fi access points located in the 6 story Bren Hall building at UC Irvine collected over a 9 month period from January to September of 2016. Each Wi-Fi trace consist of a triple ⟨AP_mac_address, device_mac_address, timestamp⟩ [1] and we use them to construct a total of 585K unique daily trajectories with 16K unique users. One challenge arising from real sensor data is the high level of noise. Users might connect to the same access point multiple times, or might cross areas covered by multiple access points. To address this, we discretized time to 10 minute intervals and set the location at each interval as the most frequent access point during that time.

Privacy Policy: We consider a user's daily trajectory as the object of privacy protection. Thus, neighboring databases under DP differ in a single user's daily trajectory on a specific day. This protects all properties of an individual within the time period of a specific day. We use access-point level policies to partition the dataset to its sensitive and non-sensitive parts. More specifically, our policies assume a sensitive set of access points (e.g., lounge or restroom) and classify as sensitive all trajectories that pass at least once through a sensitive access point. Based on this general recipe we define a policy P_ρ, that results in a non-sensitive dataset with $\rho/100$ share of non-sensitive records (e.g., P_{99} results in a dataset with 99%

[1] the device_mac_address is pseudo-anonymized to meet the IRB requirements for this study

non-sensitive daily trajectories). In our experiments we use $\rho = \{99, 90, 75, 50\}$ to capture different fractions of non-sensitive records.

4.5.2 Analysis Tasks

Classification: First we consider a classification task on the TIPPERS dataset: based on a user's daily trajectory we classify whether the user is a *resident* or a *visitor* to the building. More specifically, we learn a logistic regression (LR) model and an support vector machine (SVM) model, to predict whether a daily trajectory corresponds to a resident. Due to IRB restrictions, we do not have access to the true identities of the owners of each device_mac_addresses in the data. For that reason, to create the training data we classify as residents owners of device_mac_address that (a) visit Bren Hall at least 10 days per month for the last 5 months and either (b) work beyond 7pm once a week, or (c) work more than 6 hours at once per week; every other user is classified as a visitor. In total, we detected 381 residents that correspond to $\sim 43K$ daily trajectories (out of $\sim 553K$ total daily trajectories). We use the following features derived from the daily trajectories: duration of stay at Bren Hall, distinct number of access points visited during a day, and the number of times each individual access point is visited in a trajectory. We also consider frequent patterns of the form (AP_1, AP_2, AP_3) from the trajectories. A trajectory satisfies a frequent pattern if it visits the three access points during the day at consecutive time intervals. We chose patterns that appear in at least 50 trajectories and construct a feature for each pattern using the number of times the pattern appears in a daily trajectory.

We evaluate the performance of all LR classifiers using the receiver operating characteristic (ROC) curve, that is created by plotting the true positive rate versus the false positive rate for multiple thresholds over the probability of predicting a test record as belonging to a resident. The accuracy is usually reported using the average area under the ROC curve

(AUC) over 10-fold cross validation, which is a value between [0,1]. We report $1.0 - \text{AUC}$ as a measure of prediction error.

High Dimensional Histograms: We also consider the task of privately releasing high dimensional histograms with high sensitivity. Using the TIPPERS dataset, we want to release the number of distinct users for every n consecutive access points in a trajectory (viz., the n-gram). The challenges of this task are twofold. First the domain size for the histogram is 64^n. Second, the histogram over all bins has sensitivity 64^n since a user might appear in all n-grams. We use the mean relative error (MRE) to measure the quality of our private n-gram estimates. More specifically, for a histogram $\mathbf{x}$ of size d and its private estimate $\tilde{\mathbf{x}}$, we have:

$$\text{MRE}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{1}{d} \sum_{i=1}^{d} |x_i - \tilde{x}_i| / \max(x_i, \delta)$$

In our experiments, we set $\delta = 1$ and we report the average MRE over independent executions.

Low Dimensional Histograms: We also evaluate our algorithms on releasing 2-D histograms under OSP. We used TIPPERS to generate a 2-D histogram that contains the number of distinct users that connected to every access point and for every hour of a single day.

4.5.3 Experimental Results

Classification

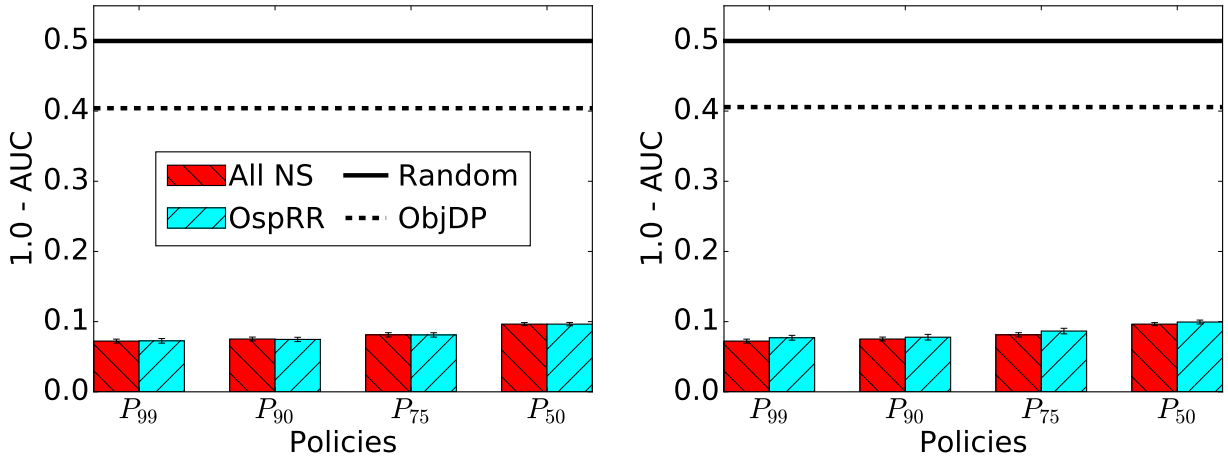
Algorithms: We compare four algorithms.

- *All NS* is a *baseline that does not satisfy OSP* that runs a non-private classifier on all the non-sensitive records. This captures the threshold algorithm that satisfies personalized DP [32]. Recall that personalized DP is susceptible to the exclusion attack.

- *OspRR* is our OSP strategy of releasing true non-sensitive records using OspRR and then running a non-private classifier.

- *ObjDP* is an implementation of the objective perturbation technique for differentially private empirical risk minimization [14]. As prescribed by the authors, we normalized feature vectors to ensure the norm is bounded by 1 before running the method on our dataset.

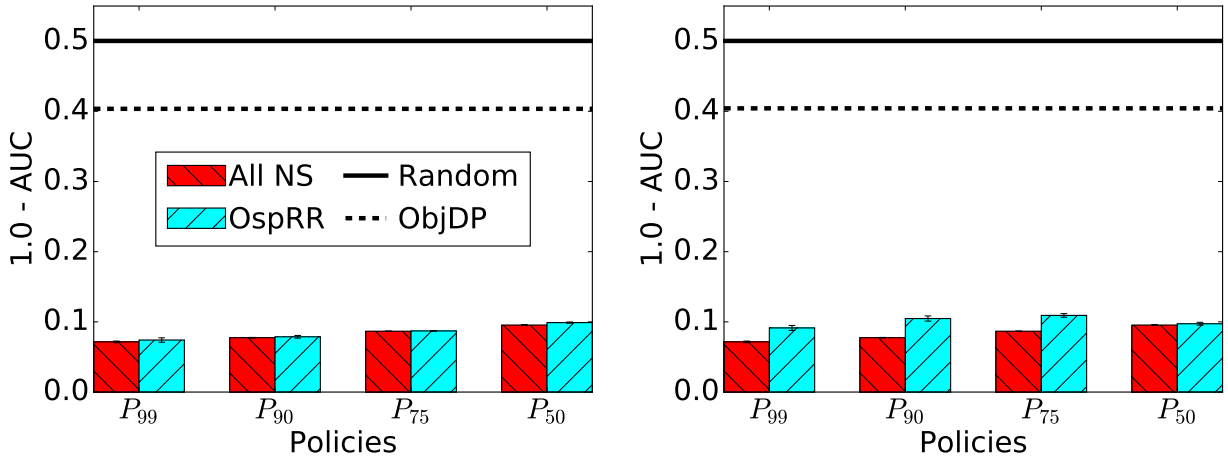
- *Random* is a baseline that randomly predicts a label based on just the label distribution (and ignoring all the features).



(a) $\epsilon = 1$ (b) $\epsilon = 0.1$

Figure 4.1: 1.0−AUC metric for residents classification using LR model.

Results: In Figures 4.1, 4.2, we report the error (1.0−AUC) for the policies described in Section 4.5.1 for two levels of privacy budget, $\epsilon = 1.0$ and $\epsilon = 0.1$. In each figure, we plot All NS and OspRR using bars, as their accuracy differs based on the fraction of non-sensitive records in the dataset. We report the error of ObjDP and Random using a straight line. We see that OspRR achieves as low an error as the non-private baseline All NS, and the absolute



(a) $\epsilon = 1$ (b) $\epsilon = 0.1$

Figure 4.2: 1.0−AUC metric for residents classification using SVM model.

errors are close to 10%. As expected, as the fraction of non-sensitive records decreases the error increases (as the number of records used by both All NS and OspRR to train the LR model decreases). ObjDP has high error close to that of the Random baseline. This shows that considering all the records as sensitive results in a very high cost in terms of utility, and OSP algorithms are able to leverage non-sensitive records to alleviate this problem.

High Dimensional Histograms

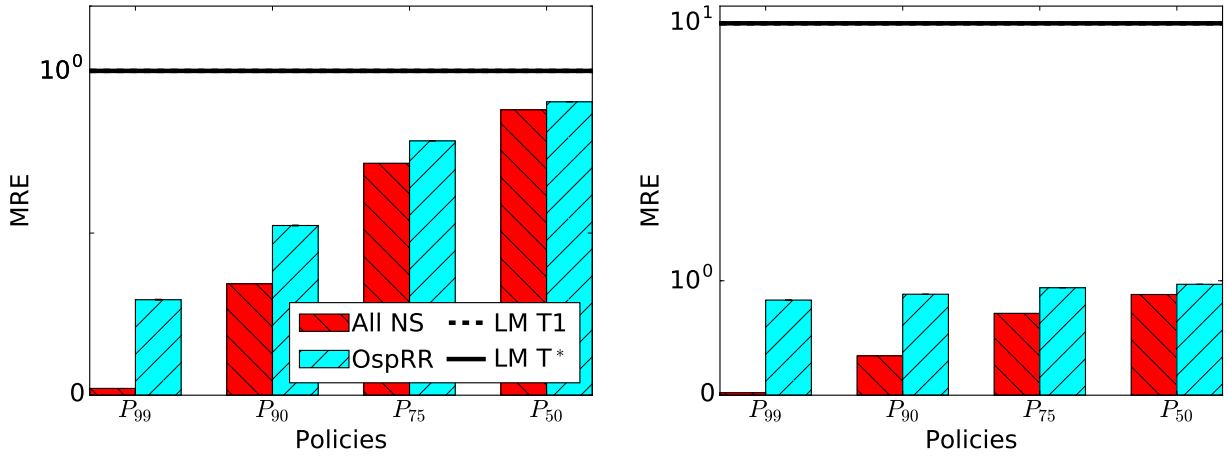
Algorithms: As before, All NS and OspRR represent our non-private and OSP algorithms, respectively, that release non-sensitive records, which are then used to answer the high-dimensional n-gram counting task (with no noise added). Under DP, the n-gram counting task has high sensitivity. So, we use *truncation* [34], and choose at most k n-grams from each daily trajectory, thus reducing the sensitivity of releasing the histogram to $2 \cdot k$. Since most of the n-grams are expected to have small or zero frequency, we added an extra post-processing step that converts all negative counts to zero. We report the mean relative error (MRE) of the Laplace mechanism with truncation of $k = 1$ (denoted by LM T1). We also report the MRE of Laplace mechanism with the optimal choice of k for truncation (denoted by LM

T*). LM T* does not satisfy DP, since the optimal k is chosen by running the algorithm for all k. But, even given this advantage LM T* performs poorly. Given the high domain size, it was intractable to materialize and perturb the complete histogram for LM when n is large. In our experiments, we only materialized the non-zero counts, and we analytically computed the contribution of the zero counts to MRE. We used the following formula to estimate the total MRE.

$$\text{MRE}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{1}{d} \sum_{i:x_i>0} |x_i - \tilde{x}_i| / \max(x_i, \delta) + \frac{1}{d} \sum_{i:x_i=0} \frac{1}{\epsilon \cdot \delta}$$

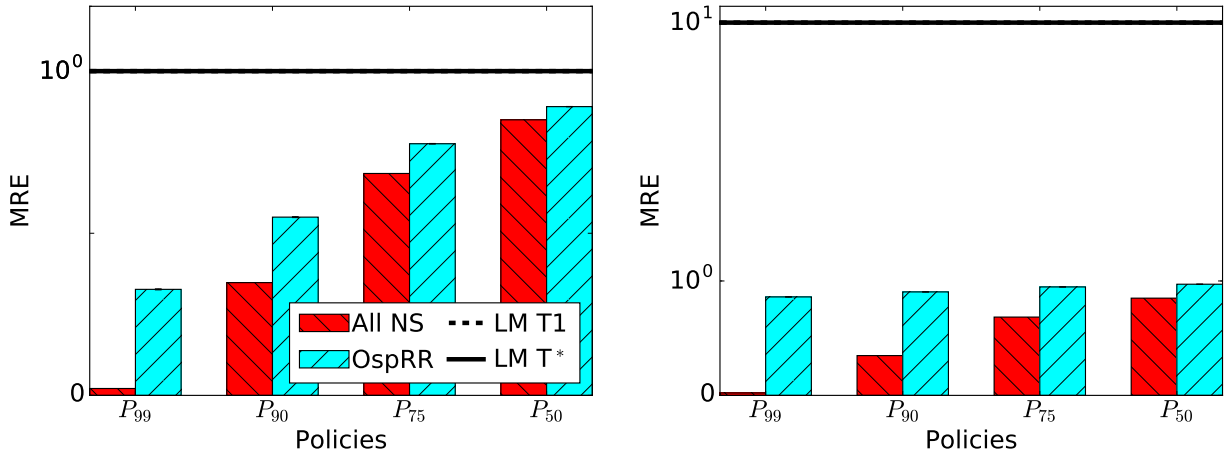
where $1/\epsilon$, is the expected L_1 error when the true frequency is zero. This problem does not arise in case of OspRR, because it always returns a subset of the non-zero n-grams.

Results: In Figures 4.3, 4.4, we plot the MRE error of the 4-grams and 5-grams histograms for four different values of the fraction of non-sensitive records in the data. We report All NS and OspRR error using bars, and the errors of LM T1 and LM T* using straight lines (since the DP algorithms' error is independent of the fraction of non-sensitive values). We observed that the optimal choice of truncation parameter k for the 4- and 5-gram counting task was $k = 1$. We see that All NS has error smaller than OspRR, but the difference is not too large when there are 10-50% sensitive records in the dataset. When $\epsilon = 1$, LM algorithms has error comparable to OspRR, when 50% of the records are non-sensitive. For smaller privacy budgets, i.e. 0.1, the LM algorithms have an order of magnitude poorer performance than the OspRR algorithm. Thus, we see again that leveraging the presence of non-sensitive records can result in significant utility gains. We did not consider more sophisticated DP techniques for releasing high-dimensional histograms. We believe their error will also be an order of magnitude larger than the OSP algorithm, since they will also have to contend with the high sensitivity and would need to use truncation.



(a) $\epsilon = 1$ (b) $\epsilon = 0.1$

Figure 4.3: Mean Relative Error 4-grams



(a) $\epsilon = 1$ (b) $\epsilon = 0.1$

Figure 4.4: Mean Relative Error 5-grams

Low Dimensional Histograms

Algorithms: For the task of releasing low dimensional histograms, we compare OspLaplaceL1 mechanism with OspRR, and DAWA, the state of the art DP algorithm for this task according to a recent benchmark study [27].

Results: TIPPERS Figure 4.5 reports the mean relative error and Figures 4.6a and 4.6b report the median and 95% per bin relative error. Overall we see that across error metrics,

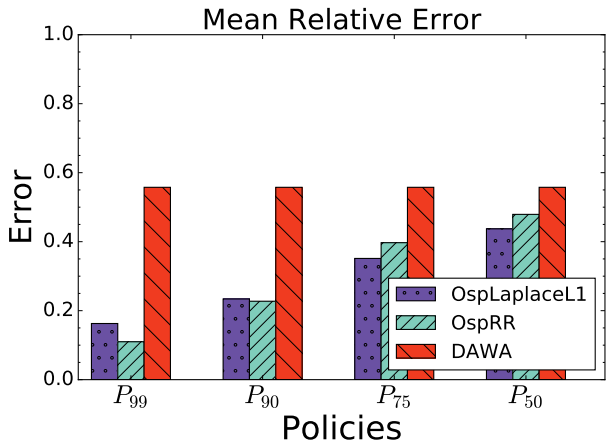
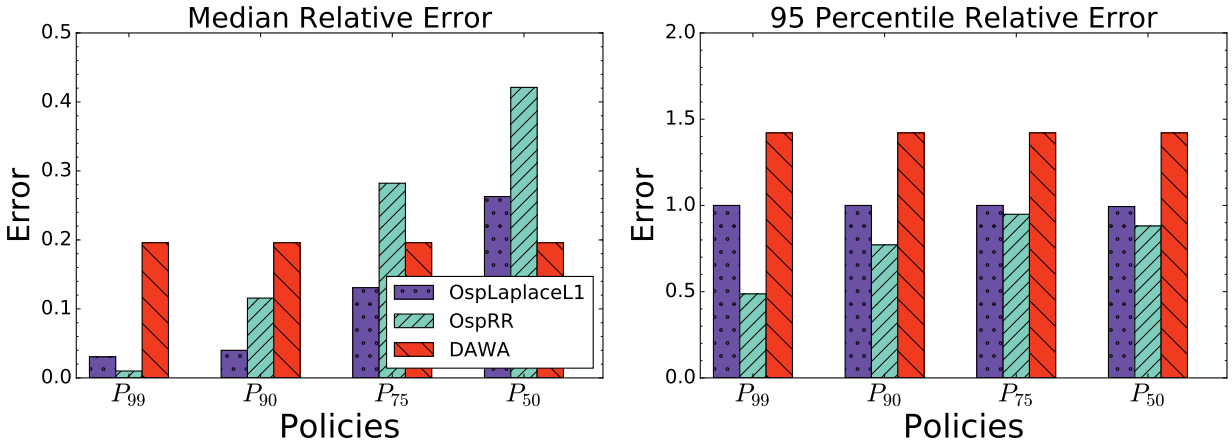


Figure 4.5: Mean Relative Error on the TIPPERS histogram.



(a) REL_{50} (b) REL_{95}

Figure 4.6: Relative per Bin Error for the TIPPERS histogram at α percentile.

OSP algorithms offer significant improvements over DAWA for all fractions of non-sensitive values. Even when only half the data records are non-sensitive we see that OSP algorithms achieve better error rates in MRE and REL_{95}, which means that the OSP algorithms have an advantage in reporting counts of bins with otherwise high error.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis, we proposed solution for some of the challenges that arise in the context of privacy preserving data sharing.

In Chapter 3, we introduced SORTaki a framework for incorporating sorting with any existing or future data dependent histogramming technique. We developed a new *finalizer* and proposed a scalable dynamic programming based *partitioner*, that further improve sorting based techniques over the state of the art. We performed a thorough and principled evaluation of sorting enhanced algorithms. Our experiments showed that sorting can help the most in case of small workloads and datasets with large domains, big scales and "random" shapes.

In Chapter 4, we presented a new privacy definition called one-sided privacy that allows for different levels of privacy guarantees (sensitive or non-sensitive) to be applied to different records in the dataset. The level of protection is determined by the value of a record, and

hence, whether a record is sensitive or non-sensitive must also be hidden. This makes the privacy definition unique. We present the fundamental theory behind the privacy definition, compare it to existing definitions and present input and output perturbation algorithms. We present case studies where one-sided privacy can be used, along with effect of our algorithms on the utility of analyzing the data in these scenarios.

5.2 Future Work

In this thesis, we only solved some of the problems that appear in privacy preserving data sharing. There are several interesting directions for future investigation.

Sorting is a very powerful method for improving the quality of DP histograms. A crucial next step is to make sorting an out of the box solution. We envision that the right path would be to incorporate the conclusions of our experiments into meta-algorithms like Pythia [39]. Additionally, an interesting direction is to create a technique that releases the sort order directly while satisfying the DP-criterion.

One-sided privacy (OSP) presents a new direction for privacy research in that it is the first formal privacy definition that exploits the sensitive/non-sensitive dichotomy of data to allow true data to be released with formal privacy guarantees. This thesis focused on motivating and defining OSP and highlighting its advantages both in the context of data sharing tasks where differential privacy (DP) either does not apply or offers very low utility, as well as, in situations where DP performs well, but OSP appropriately combined with DP offers significant improvement. Below we highlight some of the key extensions that we believe offer interesting directions of future work.

Levels of privacy protection. One interesting direction is to extend the policies to classify data into not just sensitive and non-sensitive, but into multiple levels of sensitivity. For

instance, under our current definition records that have $P(r) = 0$ require privacy at the level of ϵ, and $P(r) = 1$ do not require privacy ($\epsilon = \infty$). One could consider a scenario with non-sensitive records have a finite but larger $\epsilon' > \epsilon$. A more general definition would be to define P with a larger range; i.e., $P(r)$ could take values in $\{0, 1, \ldots, k\}$, and the definition requires records with $P(r) = i$ have privacy at the level of ϵ_i, where $\epsilon_0 \leq \epsilon_1 \leq \ldots \epsilon_k = \infty$. Extending our algorithms to capture these more general one-sided policies is an interesting avenue for future work.

Policies that go beyond records. In this thesis, we considered policy functions that define whether or not a specific record is non-sensitive. One could extend this to the case where a policy specifies a view over the database that is non-sensitive (like in the work on authorization views [56]), or a policy that determines that a view is sensitive (negative authorization [10]). While there has been work on whether or not a query leaks any information about a view [45, 49], there is no work that allows some bounded information to be leaked (like in DP) about the sensitive records in such a setting.

One-sided privacy and constraints. Theorem 4.1 showed that OSP algorithms ensure freedom from exlusion attacks as long as an adversary believes the target record is independent of the rest of the database. As in the field of differentially private data analysis problems arise when the records in the database are correlated. For instance, in a location privacy setting, a specific non-sensitive location may be reachable only through a set of locations that are all sensitive. Revealing the fact that a user was in that location (even using our randomized response algorithm) will reveal the fact that the user was in a sensitive location in the previous time point with certainty, thus resulting in a privacy breach. Designing algorithms for one-sided privacy in the presence of constraints in an important and interesting avenue for future work.

Policy Specification and Enforcement OSP definition and mechanisms assume the presence of an explicit policy function that partitions data into sensitive nad non-sensitive classes.

As we discussed in the introduction of Chapter 4, there are several application domains where explicit privacy policies and/or regulations exist that naturally lead to such a classification. Furthermore, good practice followed by several data collectors (e.g., e-commerce sites, search engines, etc.) empowers users to opt-in or opt-out. Such a practice as well lends itself directly to explicit policy function. Nonetheless, if we are to use OSP in general application context, mechanisms to specify comprehensive policies that dictate data sensitivity or, better still, learn such policies, perhaps through appropriate machine learning techniques offers rich fertile area for future research. Finally, our focus in this work has been on appropriate privacy definition and mechanisms to realize OSP. We have not focused on efficient approach to implement OSP, especially in online setting wherein user's may dynamically ask queries.

Bibliography

[1] Tippers: A live IoT testbed. `http://tippersweb.ics.uci.edu/`.

[2] M. S. Ackerman, L. F. Cranor, and J. Reagle. Privacy in e-commerce: Examining user scenarios and privacy preferences. In *ACM EC*, EC '99, pages 1–8, New York, NY, USA, 1999. ACM.

[3] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *IEEE Security and Privacy*, 3(1):26–33, Jan. 2005.

[4] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012.

[5] C. C. Aggarwal and P. S. Yu, editors. *Privacy-Preserving Data Mining - Models and Algorithms*, volume 34 of *Advances in Database Systems*. Springer, 2008.

[6] M. Alaggan, S. Gambs, , and A.-M. . Kermarrec. Heterogeneous differential privacy. *Journal of Privacy and Confidentiality*, 7(6), 2017.

[7] M. Barbaro and T. Z. Jr. A face is exposed for aol searcher no. 4417749. `http://www.nytimes.com/2006/08/09/technology/09aol.html`, 2006.

[8] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor. Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs. *Personal Ubiquitous Comput.*, 15(7):679–694, Oct. 2011.

[9] B. Berendt, O. Günther, and S. Spiekermann. Privacy in e-commerce: Stated preferences vs. actual behavior. *Commun. ACM*, 48(4):101–106, Apr. 2005.

[10] E. Bertino, S. Jajodia, and P. Samarati. A flexible authorization mechanism for relational data management systems. *ACM Trans. Inf. Syst.*, 17(2):101–140, Apr. 1999.

[11] K. Browder and M. Davidson. The virtual private database in oracle9ir2. *Oracle Technical White Paper, Oracle Corporation*, 500, 2002.

[12] G. M. Chao Li, Michael Hay and Y. Wang. A data- and workload-aware query answering algorithm for range queries under differential privacy. In *VLDB*, pages 341–352, 2014.

[13] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In *NIPS*, pages 289–296, 2008.

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

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

[16] G. Cormode, C. M. Procopiuc, D. Srivastava, and G. Yaroslavtsev. Accurate and efficient private release of datacubes and contingency tables. *arXiv*, arXiv:1207.6096v1, 2012.

[17] I. Corporation. Smart buildings with internet of things technologies. `https://www.intel.com/content/www/us/en/smart-buildings/overview.html`, 2017.

[18] W.-Y. Day, N. Li, and M. Lyu. Publishing graph degree distribution with node differential privacy. In *ACM SIGMOD*, 2016.

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

[20] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006.

[21] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, Aug. 2014.

[22] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[23] S. Haney, A. Machanavajjhala, and B. Ding. Design of policy-aware differentially private algorithms. *PVLDB*, 9(4), 2015.

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

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

[26] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 139–154, New York, NY, USA, 2016. ACM.

[27] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, 2016.

[28] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, pages 1021–1032, 2010.

[29] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially-private queries through consistency. In *PVLDB*, pages 1021–1032, 2010.

[30] X. He, A. Machanavajjhala, and B. Ding. Blowfish privacy: tuning privacy-utility trade-offs using policies. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1447–1458, 2014.

[31] B. Howard, M. G. Robert, and B. F. Phil. Data sharing, an ethical and scientific imperative, March 2016.

[32] Z. Jorgensen, T. Yu, and G. Cormode. Conservative or liberal? personalized differential privacy. In *ICDE 2015*, pages 1023–1034, 2015.

[33] V. Karwa, S. Raskhodnikova, and A. S. G. Yaroslavtsev. Private analysis of graph structure. *PVLDB*, 4(11):1146–1157, 2011.

[34] S. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *TCC*, 2013.

[35] G. Kellaris and S. Papadopoulos. Practical differential privacy via grouping and smoothing. In *PVLDB*, 2013.

[36] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM.

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

[38] D. Kifer and A. Machanavajjhala. A rigorous and customizable framework for privacy. In *PODS*, pages 77–88, 2012.

[39] I. Kotsogiannis, A. Machanavajjhala, M. Hay, and G. Miklau. Pythia: Data dependent differentially private algorithm selection. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1323–1337, New York, NY, USA, 2017. ACM.

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

[41] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *Proceedings of the VLDB Endowment*, 7(5):341–352, 2014.

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

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

[44] L. Li, K. Zhou, G.-R. Xue, H. Zha, and Y. Yu. Enhancing diversity, coverage and balance for summarization through structure learning. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 71–80, New York, NY, USA, 2009. ACM.

[45] A. Machanavajjhala and J. Gehrke. On the efficiency of checking perfect privacy. In *PODS '06*, PODS '06, pages 163–172, New York, NY, USA, 2006. ACM.

[46] G. Maldoff. Top 10 operational impacts of the gdpr: Part3 - consent. `https://iapp.org/news/a/top-10-operational-impacts-of-the-gdpr-part-3-consent/`, 2017.

[47] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the netflix prize contenders. In *KDD*, pages 627–636, 2009.

[48] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[49] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD '04*, SIGMOD '04, pages 575–586, New York, NY, USA, 2004. ACM.

[50] G. R. Milne and A. J. Rohm. Consumer privacy and name removal across direct marketing channels: Exploring opt-in and opt-out alternatives. *Journal of Public Policy & Marketing*, 19(2):238–249, 2000.

[51] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[52] E. Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[53] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD '96*, pages 294–305, 1996.

[54] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, pages 1954–1965, 2013.

[55] W. Qardaji, W. Yang, and N. Li. Priview: Practical differentially private release of marginal contingency tables. In *SIGMOD '14*, pages 1435–1446, 2014.

[56] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD Conference 2004*, SIGMOD '04, pages 551–562, New York, NY, USA, 2004. ACM.

[57] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[58] R. Singel. Netflix cancels recommendation contet after privacy lawsuit. `https://www.wired.com/2010/03/netflix-cancels-contest/`, 2010.

[59] R. Steven, G. Katie, G. Ronald, G. Josiah, and S. Matthew. Integrated public use microdata series: Version 6.0. http://doi.org/10.18128/D010.V6.0, 2015.

[60] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[61] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. on Knowl. and Data Eng.*, pages 1200–1214, 2011.

[62] Y. Xiao, J. Gardner, and L. Xiong. Dpcube: Releasing differentially private data cubes for health information. In *ICDE*, pages 1305–1308. IEEE, 2012.

[63] Y. Xiao, J. J. Gardner, and L. Xiong. Dpcube: Releasing differentially private data cubes for health information. In *ICDE*, pages 1305–1308, 2012.

[64] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. In *Secure Data Management*, pages 150–168, 2010.

[65] J. Xu, D. V. Kalashnikov, and S. Mehrotra. Efficient summarization framework for multi-attribute uncertain data. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 421–432, 2014.

[66] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *ICDE*, pages 32–43, 2012.

[67] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *PVLDB*, pages 797–822, 2013.

[68] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *PVLDB*, 2012.

[69] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 587–595. SIAM, 2014.