ANALYZING SENSITIVE DATA WITH LOCAL DIFFERENTIAL PRIVACY

by

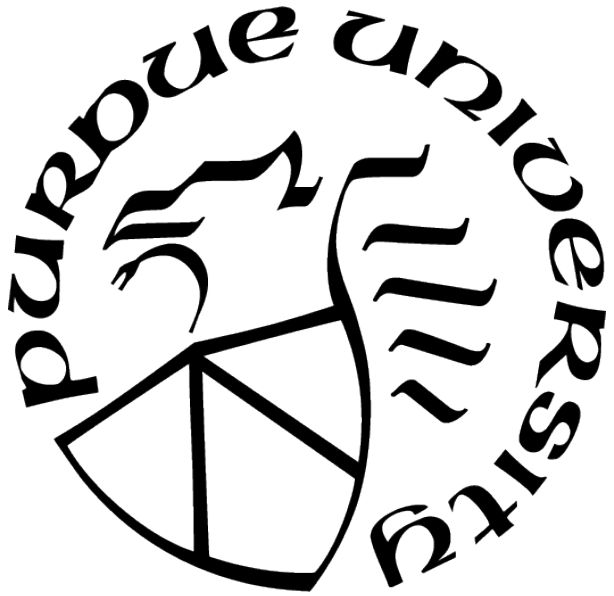
Tianhao Wang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



Department of Computer Science

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Ninghui Li, Chair

Department of Computer Science

Dr. Somesh Jha

Department of Computer Science, University of Wisconsin-Madison

Dr. Elisa Bertino

Department of Computer Science

Dr. Jeremiah Blocki

Department of Computer Science

Dr. Jean Honorio

Department of Computer Science

Approved by:

Dr. Kihong Park

ACKNOWLEDGMENTS

First and foremost, I would like to express my greatest thanks to my advisor, Professor Ninghui Li, for his continued guidance, support, and encouragement during my Ph.D. study. I am so fortunate to be leaded into the area of data privacy by him. I am truly grateful for his vision and direction about research, inspiration, and perfect personality as advisor.

Also I would like to show my deep gratitude to other doctoral committee members, Professor Somesh Jha, Professor Elisa Bertino, and Professor Jeremiah Blocki, Jean Honorio for their invaluable help on my research and constructive suggestions on the dissertation.

Many thanks to my internship mentors, Dr. Bolin Ding at Alibaba and Professor Ashwin Machanavajjhala at Tumult Labs and Duke University. I learned a lot from them and gained a lot of valuable experiences, which are important to my Ph.D. research.

I also thank Michael Backes, Sze Yiu Chau, Joann Chen, Min Chen, Yueqiang Cheng, Omar Chowdhury, Fabrizio Cicala, Graham Cormode, Huangyi Ge, Victor Gonsalves, Cheng Hong, Zhicong Huang, Mathias Humbert, Tejas Kulkarni, Huian Li, Zhou Li, Faming Liang, Milan Lopuhaä-Zwakenberg, Hemanta Maji, Boris Skoric, Divesh Srivastava, Dong Su, Weicheng Wang, Aiping Xiong, Min Xu, Jianyu Yang, Yang Yang, Yang Zhang, Zhikun Zhang, Yunlei Zhao, Jingren Zhou, Xukai Zou for enjoyable collaborations over the years.

Last but not least, my heartfelt appreciation goes to my whole family. I can always feel their love, inspiration, bless, and support behind me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Vast amounts of sensitive personal information are collected by companies, institutions and governments. A key technological challenge is how to effectively extract knowledge from data while preserving the privacy of the individuals involved. In this dissertation, we address this challenge from the perspective of privacy-preserving data collection and analysis. We focus on investigation of a technique called local differential privacy (LDP) and studied several aspects of it.

In particular, the thesis serves as a comprehensive study of multiple aspects of the LDP field. We investigated the following seven problems: **(1)** We studied LDP primitives, i.e., the basic mechanisms that are used to build LDP protocols. **(2)** We then studied the problem when the domain size is very big (e.g., larger than 2^{32}), where finding the values with high frequency is a challenge, because one needs to enumerate through all values. **(3)** Another interesting setting is when each user possesses a set of values, instead of a single private value. **(4)** With the basic problems visited, we then aim to make the LDP protocols practical for real-world scenarios. We investigated the case where each user's data is high-dimensional (e.g., in the census survey, each user has multiple questions to answer), and the goal is to recover the joint distribution among the attributes. **(5)** We also built a system for companies to issue SQL queries over the data protected under LDP, where each user is associated with some public weights and holds some private values; an LDP version of the values is sent to the server from each user. **(6)** To further increase the accuracy of LDP, we study how to add post-processing steps to protocols to make them consistent while achieving high accuracy for a wide range of tasks, including frequencies of individual values, frequencies of the most frequent values, and frequencies of subsets of values. **(7)** Finally, we investigate a different model of LDP which is called the shuffler model. While users still use LDP algorithms to report their sensitive data, now there exists a semi-trusted shuffler that shuffles the users' reports and then send them to the server. This model provides better utility but at the cost of requiring more trust that the shuffler should not collude with the server.

1. INTRODUCTION

Large volumes of users' data about their profiles and activities are collected by enterprises to make informed business decisions. In order to meet users' expectation of their privacy, applications and services must provide rigorous privacy guarantees on how their data is collected and analyzed. Differential privacy (DP) [1] has emerged as the *de facto* standard for privacy guarantees. Recently, techniques for satisfying DP in the local setting, which we call LDP, have been studied deployed by, e.g., Google [2], Apple [3], and Microsoft [4].

Together with my advisor and colleagues, we studied LDP primitives, i.e., the basic mechanisms that are used to build LDP protocols. Specifically, assuming that each user has a private value from a known domain, these primitives are able to estimate the distribution of values, thus are also called frequency oracles. In Chapter 3, we introduce a framework that generalizes several frequency oracles proposed in the literature. Our in-depth analysis enables us to choose optimal parameters, resulting in two new protocols. The estimation error of the proposed protocols is 1/14 that of Apple's implementation, and 1/2 that of Google's RAPPOR.

Frequency oracles can handle domains of limited size. When the domain size is very big (e.g., larger than 2^{32}), finding the values with high frequency (referred to as the heavy hitters) is a challenge, because one needs to enumerate through all values. In Chapter 4, we proposed protocols to find the heavy hitters efficiently. The high-level idea is to iteratively identify increasingly longer frequent prefixes (assuming the values are represented by binary strings). With a thorough analysis of the utility, we found that within the computational limit, it works best to make as few iterations as possible. This design makes our protocol effectively find around 2× more heavy hitters than existing methods.

Another interesting setting is when each user possesses a set of values, instead of a single private value. In this setting, an additional padding and sampling step is needed to find the frequent values and estimate their frequencies. In Chapter 5, we formally defined such padding and sample based frequency oracles; and we identified the privacy amplification property. As a result, compared with existing methods, our protocol can find 3× more frequent items, with 3 orders of magnitudes less estimation errors.

With the basic problems visited, we then aim to make the LDP protocols practical for real-world scenarios. In Chapter 6, we investigated the case where each user's data is high-dimensional (e.g., in the census survey, each user has multiple questions to answer), and the goal is to recover the joint distribution among the attributes. The high level idea is that, as there is a limit on accuracy, working on all possible joint distributions will make each one less accurate; therefore, we focus on some joint distributions, and use these few but accurate estimations to synthesize the uncovered ones. As a result, our proposal consistently performs at least one magnitudes better than existing protocols.

In the traditional model of LDP (and all centralized DP models), it is assumed that all the information associated with the user is private and should be protected. This is not the case in many practical scenarios of local DP. As each user reports to the server, there are internally public information associated with the user that is known to the server; and such information is not supposed to be protected by LDP. In Chapter 7, we utilize this information and build a system for companies to issue SQL queries over the data protected under LDP, where each user is associated with some public weights and holds some private values; an LDP version of the values is sent to the server from each user.

To further increase the accuracy of LDP, we study how to add post-processing steps to primitives to make them consistent while achieving high accuracy for a wide range of tasks, including frequencies of individual values, frequencies of the most frequent values, and frequencies of subsets of values. In Chapter 8, we consider 10 different methods, some of them explicitly proposed before in the literature, and others introduced in this paper. We establish theoretical relationships between some of them and conducted extensive experimental evaluations to understand which methods should be used for different query tasks.

Finally, in Chapter 9, we study a variant of LDP that introduces an intermediate server with the assumption that this intermediate server does not collude with the aggregator. Under this assumption, less noise can be added to achieve the same privacy guarantee as LDP, thus improving utility for the data collection task. We analyze the system model and identify potential adversaries. We then make two improvements: a new algorithm that achieves a better privacy-utility tradeoff; and a novel protocol that provides better protection against various attacks.

2. BACKGROUND

We consider a setting where there are several *users* and one *aggregator*. Each user possesses a value v from a domain D, and the aggregator wants to learn the distribution of values among all users, in a way that protects the privacy of individual users.

2.1 Differential Privacy

Definition 2.1.1 (ϵ-Differential Privacy)**.** *An algorithm $\mathcal{A}(\cdot)$ satisfies ϵ-differential privacy (ϵ-DP) if and only if for any two **neighboring** datasets D and D', we have*

$$\forall t \in Range(\mathcal{A}) : \Pr[\mathcal{A}(D) = t] \leq e^{\epsilon} \Pr[\mathcal{A}(D') = t].$$

where $Range(\mathcal{A})$ denotes the set of all possible outputs of the algorithm $\mathcal{A}$.

Here neighboring is the bounded definition, which means two datasets D and D' differ by replacing one element (instead of the unbounded definition where one of D or D' is obtained by inserting or deleting one element from the other).

2.2 Differential Privacy in the Local Setting

In the local setting, each user perturbs the input value v using an algorithm $\mathcal{A}_L$ and sends $\mathcal{A}_L(v)$ to the aggregator. The formal privacy requirement is that the algorithm $\mathcal{A}_L(\cdot)$ satisfies the following property:

Definition 2.2.1 (ϵ-Local Differential Privacy)**.** *An algorithm $\mathcal{A}_L(\cdot)$ satisfies ϵ-local differential privacy (ϵ-LDP) if and only if for any input $v_1, v_2 \in D$, we have*

$$\forall T \subseteq Range(\mathcal{A}_L) : \Pr[\mathcal{A}_L(v_1) \in T] \leq e^{\epsilon} \Pr[\mathcal{A}_L(v_2) \in T],$$

where $Range(\mathcal{A}_L)$ denotes the set of all possible outputs of the algorithm $\mathcal{A}_L$.

Compared to the centralized setting, the local version of DP offers a stronger level of protection, because each user only reports the perturbed data. Each user's privacy is still protected even if the aggregator is malicious.

Given an ϵ-DP algorithm $\mathcal{A}$, we can apply it to datasets of a single item; and this also satisfies ϵ-LDP. On the other hand, given an ϵ-LDP algorithm $\mathcal{A}_L$, we can apply it to each element of the database to satisfy ϵ-DP. The difference between these two definitions is the system model.

Properties of LDP. Both notions enjoy properties of sequential composition, parallel composition, and post-processing. As we focus on LDP, the theorems are presented informally for LDP; but the DP version can be similarly derived. Specifically, (1) if the user executes a set of functions, each satisfying ϵ_i-LDP, then the whole process satisfies $\sum \epsilon_i$-LDP. The value ϵ is also called the *privacy budget*. (2) parallel composition is trivial in the local setting: if the users are partitioned into groups, each evaluating a separate ϵ-LDP algorithm, the whole process is ϵ-LDP. (3) any modification to the perturbed result do not influence the privacy guarantee of LDP.

3. FREQUENCY ORACLE

(A version of this chapter has been previously published in USENIX Security 2017 [5]. There is also a public version [6] with some additional information.)

In this chapter, we focus on frequency estimation. This is the most basic primitive and is a necessary building block for other goals in the subsequent sections. Improving this will improve effectiveness of other protocols.

We assume there are n users. Each user j has one value v^j, which can be viewed as the user's answer to a given question, and reports once. We use d to denote the size of the domain of the values the users have, and $D = [d]$ to denote the set $\{1, 2, \ldots, d\}$.

The most basic goal of the server is **frequency estimation**, i.e., estimate, for a given value $v \in [d]$, how many users have the value v. Such a data collection protocol consists of the following algorithms:

- Encode is executed by each user. The algorithm takes an input value v and outputs an encoded value x.
- Perturb, which takes an encoded value x and outputs y. Each user with value v reports $y = \mathsf{Perturb}(\mathsf{Encode}(v))$. For compactness, we use $\mathsf{PE}(\cdot)$ to denote the composition of the encoding and perturbation algorithms, i.e., $\mathsf{PE}(\cdot) = \mathsf{Perturb}(\mathsf{Encode}(\cdot))$. $\mathsf{PE}(\cdot)$ should satisfy ϵ-LDP given in Definition 7.1.1.
- Aggregate is executed by the aggregator; it takes all the reported values, and outputs aggregated information.

3.1 Existing Work

3.1.1 Basic Rappor

Rappor [2] is designed to enable longitudinal collections, where the collection happens multiple times. Indeed, Chrome's implementation of Rappor [7] collects answers to some

questions once every 30 minutes. Two protocols, Basic Rappor and Rappor, are proposed in [2]. We first describe Basic Rappor.

Encoding. $\mathsf{Encode}(v) = B_0$, where B_0 is a length-d binary vector such that $B_0[v] = 1$ and $B_0[i] = 0$ for $i \neq v$. We call this **Unary Encoding**.

Perturbation. $\mathsf{Perturb}(B_0)$ consists of two steps:

Step 1: Permanent randomized response: Generate B_1 such that:

$$\Pr[B_1[i] = 1] = \begin{cases} 1 - \frac{1}{2}f, & \text{if } B_0[i] = 1, \\ \frac{1}{2}f, & \text{if } B_0[i] = 0. \end{cases}$$

Rappor's implementation uses $f = 1/2$ and $f = 1/4$. Note that this randomization is **symmetric** in the sense that $\Pr[B_1[i] = 1 | B_0[i] = 1] = \Pr[B_1[i] = 0 | B_0[i] = 0] = 1 - \frac{1}{2}f$; that is, the probability that a bit of 1 is preserved equals the probability that a bit of 0 is preserved. This step is carried out only once for each value v that the user has.

Step 2: Instantaneous randomized response: Report B_2 such that:

$$\Pr[B_2[i] = 1] = \begin{cases} p, & \text{if } B_1[i] = 1, \\ q, & \text{if } B_1[i] = 0. \end{cases}$$

This step is carried out each time a user reports the value. That is, B_1 will be perturbed to generate different B_2's for each reporting. Rappor's implementation [7] uses $p = 0.75$ and $q = 0.25$, and is hence also symmetric because $p + q = 1$.

We note that as both steps are symmetric, their combined effect can also be modeled by a symmetric randomization. Moreover, we study the problem where each user only reports once. Thus without loss of generality, we ignore the instantaneous randomized response step and consider only the permanent randomized response when trying to identify effective protocols.

This step is introduced to help prevent multiple reports from the same user to be correlated; however, the rationale of this step is questionable. First, in the LDP setting, the aggregator is the adversary against which privacy protection is needed. Since a user needs to communicate with an aggregator, then correlation can be made based on other aspects of

the communication (such as the IP address). Second, the exact quantitative degree of such protection is unclear, especially because a user's reporting may consist of answers to many questions. Third, this step affects utility (because it further perturbs the value) without improving privacy guarantee, because against an adversary who sees multiple responses, such as the aggregator, this step provides little additional privacy, since the effect of instantaneous randomization can be cancelled out. We also note that when one uses p and q values such that $p+q=1$, as in the case of the Rappor implementation [7], the combined effect of the two randomization steps can be approximated by using only the first step, but with a different f value.

Aggregation. Let B^j be the reported vector of the j-th user. Ignoring the Instantaneous randomized response step, to estimate the number of times i occurs, the aggregator computes:

$$\tilde{c}(i) = \frac{\sum_j \mathbf{1}_{i \in \{i | B^j[i]=1\}} - \frac{1}{2}fn}{1-f}$$

That is, the aggregator first counts how many time i is reported by computing $\sum_j \mathbf{1}_{i \in \{i|B^j[i]=1\}}$, which counts how many reported vectors have the i'th bit being 1, and then corrects for the effect of randomization. We use $\mathbf{1}_P$ to denote the indicator function such that: $\mathbf{1}_P = 1$ if the predicate P is true and 0 otherwise.

Cost. The communication and computing cost is $\Theta(d)$ for each user, and $\Theta(nd)$ for the aggregator.

Privacy. Against an adversary who may observe multiple transmissions, this achieves ϵ-LDP for $\epsilon = \ln\left(\left(\frac{1-\frac{1}{2}f}{\frac{1}{2}f}\right)^2\right)$, which is $\ln 9$ for $f = 1/2$ and $\ln 49$ for $f = 1/4$.

3.1.2 Rappor

Basic Rappor uses unary encoding, and does not scale when d is large. To address this problem, Rappor uses Bloom filters. While Bloom filters are typically used to encode a set for membership testing, in Rappor it is used to encode a single element.

Encoding. Encoding uses a set of m hash functions $\mathbb{H} = \{H_1, H_2, \ldots, H_m\}$, each of which outputs an integer in $[k] = \{0, 1, \ldots, k-1\}$. $\mathsf{Encode}(v) = B_0$, which is k-bit binary vector such that

$$B_0[i] = \begin{cases} 1, & \text{if} \quad \exists H \in \mathbb{H}, s.t., H(v) = i, \\ 0, & \text{otherwise.} \end{cases}$$

Perturbation. The perturbation process is identical to that of Basic Rappor.

Aggregation. The use of shared hashing creates challenges due to potential collisions. If two values happen to be hashed to the same set of indices, it becomes impossible to distinguish them. To deal with this problem, Rappor introduces the concept of cohorts. The users are divided into a number of cohorts. Each cohort uses a different set of hash functions, so that the effect of collisions is limited to within one cohort. However, partial collisions, i.e., two values are hashed to overlapping (though not identical) sets of indices, can still occur and interfere with estimation. These complexities make the aggregation algorithm more complicated. Rappor uses LASSO and linear regression to estimate frequencies of values.

Cost. The communication and computing cost is $\Theta(k)$ for each user. The aggregator's computation cost is higher than Basic Rappor due to the usage of LASSO and regression.

Privacy. Rappor achieves ϵ-LDP for $\epsilon = \ln\left(\left(\frac{1-\frac{1}{2}f}{\frac{1}{2}f}\right)^{2m}\right)$. The Rappor implementation uses $m = 2$; thus this is $\ln 81 \approx 4.39$ for $f = 1/2$ and $\ln 7^4 \approx 7.78$ for $f = 1/4$.

3.1.3 Random Matrix Projection

Bassily and Smith [8] proposed a protocol that uses random matrix projection. This protocol has an additional Setup step.

Setup. The aggregator generates a public matrix $\Phi \in \{-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\}^{m \times d}$ uniformly at random. Here m is a parameter determined by the error bound, where the "error" is defined as the maximal distance between the estimation and true frequency of any domain.

Encoding. $\mathsf{Encode}(v) = \langle r, x \rangle$, where r is selected uniformly at random from $[m]$, and x is the v's element of the r's row of Φ, i.e., $x = \Phi[r, v]$.

Perturbation. $\mathsf{Perturb}(\langle r, x \rangle) = \langle r,\ b \cdot c \cdot m \cdot x \rangle$, where

$$b = \begin{cases} 1 & \text{with probability } p = \frac{e^\epsilon}{e^\epsilon + 1}, \\ -1 & \text{with probability } q = \frac{1}{e^\epsilon + 1}, \end{cases}$$

$$c = (e^\epsilon + 1)/(e^\epsilon - 1).$$

Aggregation. Given reports of the form $\langle r^j, y^j \rangle$, the estimate for $i \in [d]$ is given by

$$\tilde{c}(i) = \sum_j y^j \cdot \Phi[r^j, i].$$

The effect is that each user with input value i contributes c to $\tilde{c}(i)$ with probability p, and $-c$ with probability q; thus the expected contribution is

$$(p - q) \cdot c = \left(\frac{e^\epsilon}{e^\epsilon + 1} - \frac{1}{e^\epsilon + 1} \right) \cdot \frac{e^\epsilon + 1}{e^\epsilon - 1} = 1.$$

Because of the randomness in Φ, each user with value $\neq i$ contributes to $\tilde{c}(i)$ either c or $-c$, each with probability $1/2$; thus the expected contribution from all such users is 0. Note that each row in the matrix is essentially a random hashing function mapping each value in $[d]$

to a single bit. Each user selects such a hash function, uses it to hash her value into one bit, and then perturbs this bit using random response.

Cost. A straightforward implementation of the protocol is expensive. However, the public random matrix Φ does not need to be explicitly computed. For example, using a common pseudo-random number generator, each user can randomly choose a seed to generate a row in the matrix and send the seed in her report. With this technique, the communication cost is $\Theta(\log m)$ for each user, and the computation cost is $O(d)$ for computing one row of the Φ. The aggregator needs $\Theta(dm)$ to generate Φ, and $\Theta(md)$ to compute the estimations.

3.2 A Framework for LDP Protocols

Multiple protocols have been proposed for estimating frequencies under LDP, and one can envision other protocols. A natural research question is how do they compare with each other? Under the same level of privacy, which protocol provides better accuracy in aggregation, with lower cost? Can we come up with even better ones? To answer these questions, we define a class of LDP protocols that we call "pure".

For a protocol to be pure, we require the specification of an additional function $\mathsf{Support}$, which maps each possible output y to a set of input values that y "supports". For example, in the basic Rappor protocol, an output binary vector B is interpreted as supporting each input whose corresponding bit is 1, i.e., $\mathsf{Support}(B) = \{i \mid B[i] = 1\}$.

Definition 3.2.1 (Pure LDP Protocols)**.** *A protocol given by* PE *and* $\mathsf{Support}$ *is pure if and only if there exist two probability values* $p^* > q^*$ *such that for all* v_1*,*

$$\mathsf{Pr}\left[\mathsf{PE}(v_1) \in \{y \mid v_1 \in \mathsf{Support}(y)\}\right] = p^*,$$
$$\forall_{v_2 \neq v_1} \mathsf{Pr}\left[\mathsf{PE}(v_2) \in \{y \mid v_1 \in \mathsf{Support}(y)\}\right] = q^*.$$

A pure protocol is in some sense "pure and simple". For each input v_1, the set $\{y \mid v_1 \in \mathsf{Support}(y)\}$ identifies all outputs y that "support" v_1, and can be called the support set of v_1. A pure protocol requires the probability that any value v_1 is mapped to its own support set be the same for all values. We use p^* to denote this probability. In order to satisfy LDP,

it must be possible for a value $v_2 \neq v_1$ to be mapped to v_1's support set. It is required that this probability, which we use q^* to denote, must be the same for all pairs of v_1 and v_2. Intuitively, we want p^* to be as large as possible, and q^* to be as small as possible. However, satisfying ϵ-LDP requires that $\frac{p^*}{q^*} \leq e^\epsilon$.

Basic Rappor is pure with $p^* = 1 - \frac{f}{2}$ and $q^* = \frac{f}{2}$. Rappor is not pure because there does not exist a suitable q^* due to collisions in mapping values to bit vectors. Assuming the use of two hash functions, if v_1 is mapped to $[1, 1, 0, 0]$, v_2 is mapped to $[1, 0, 1, 0]$, and v_3 is mapped to $[0, 0, 1, 1]$, then because $[1, 1, 0, 0]$ differs from $[1, 0, 1, 0]$ by only two bits, and from $[0, 0, 1, 1]$ by four bits, the probability that v_2 is mapped to v_1's support set is higher than that of v_3 being mapped to v_1's support set.

For a pure protocol, let y^j denote the submitted value by user j, a simple aggregation technique to estimate the number of times that i occurs is as follows:

$$\tilde{c}(i) = \frac{\sum_j 1_{i \in \mathsf{Support}(y^j)} - nq^*}{p^* - q^*} \tag{3.1}$$

The intuition is that each output that supports i gives an count of 1 for i. However, this needs to be normalized, because even if every input is i, we only expect to see $n \cdot p^*$ outputs that support i, and even if input i never occurs, we expect to see $n \cdot q^*$ supports for it. Thus the original range of 0 to n is "compressed" into an expected range of nq^* to np^*. The linear transformation in Equation (3.1) corrects this effect.

Theorem 3.2.1. *For a pure LDP protocol* PE *and* Support*, Equation* (3.1) *is unbiased, i.e.,* $\forall_i \mathrm{E}[\,\tilde{c}(i)\,] = nf_i$*, where* f_i *is the fraction of times that the value* i *occurs.*

Proof:

$$\begin{aligned}
\mathrm{E}[\,\tilde{c}(i)\,] =& \mathrm{E}\left[\frac{\left(\sum_j 1_{i \in \mathsf{Support}(y^j)}\right) - nq^*}{p^* - q^*}\right] \\
=& \frac{nf_i p^* + n(1 - f_i)q^* - nq^*}{p^* - q^*} \\
=& n \cdot \frac{f_i p^* + q^* - f_i q^* - q^*}{p^* - q^*} \\
=& nf_i
\end{aligned}$$

□

The variance of the estimator in Equation (3.1) is a valuable indicator of an LDP protocol's accuracy:

Theorem 3.2.2. *For a pure LDP protocol* PE *and* Support*, the variance of the estimation* $\tilde{c}(i)$ *in Equation* (3.1) *is:*

$$\mathrm{Var}[\tilde{c}(i)] = \frac{nq^*(1-q^*)}{(p^*-q^*)^2} + \frac{nf_i(1-p^*-q^*)}{p^*-q^*} \tag{3.2}$$

PROOF: The random variable $\tilde{c}(i)$ is the (scaled) summation of n independent random variables drawn from the Bernoulli distribution. More specifically, nf_i (resp. $(1-f_i)n$) of these random variables are drawn from the Bernoulli distribution with parameter p^* (resp. q^*). Thus,

$$\begin{aligned}
\mathrm{Var}[\tilde{c}(i)] &= \mathrm{Var}\left[\frac{\left(\sum_j \mathbf{1}_{i\in\mathsf{Support}(y^j)}\right) - nq^*}{p^*-q^*}\right] \\
&= \frac{\sum_j \mathrm{Var}[\mathbf{1}_{i\in\mathsf{Support}(y^j)}]}{(p^*-q^*)^2} \\
&= \frac{nf_ip^*(1-p^*) + n(1-f_i)q^*(1-q^*)}{(p^*-q^*)^2} \\
&= \frac{nq^*(1-q^*)}{(p^*-q^*)^2} + \frac{nf_i(1-p^*-q^*)}{p^*-q^*}
\end{aligned}$$

□

In many application domains, the vast majority of values appear very infrequently, and one wants to identify the more frequent ones. The key to avoid having lots of false positives is to have low estimation variances for the infrequent values. When f_i is small, the variance in Equation (3.2) is dominated by the first term. We use Var^* to denote this approximation of the variance, that is:

$$\mathrm{Var}^*[\tilde{c}(i)] = \frac{nq^*(1-q^*)}{(p^*-q^*)^2} \tag{3.3}$$

We also note that some protocols have the property that $p^* + q^* = 1$, in which case $\text{Var}^* = \text{Var}$.

As the estimation $\tilde{c}(i)$ is the sum of many independent random variables, its distribution is very close to a normal distribution. Thus, the mean and variance of $\tilde{c}(i)$ fully characterizes the distribution of $\tilde{c}(i)$ for all practical purposes. When comparing different methods, we observe that fixing ϵ, the differences are reflected in the constants for the variance, which is where we focus our attention.

3.3 Optimizing LDP Protocols

We now cast many protocols that have been proposed into our framework of "pure" LDP protocols. Casting these protocols into the framework of pure protocols enables us to derive their variances and understand how each method's accuracy is affected by parameters such as domain size, ϵ, etc. This also enables us to generalize and optimize these protocols and propose two new protocols that improve upon existing ones. More specifically, we will consider the following protocols, which we organize by their encoding methods.

- **Direct Encoding (DE).** There is no encoding. It is a generalization of the Random Response technique (GRR).
- **Histogram Encoding (HE).** An input v is encoded as a histogram for the d possible values. The perturbation step adds noise from the Laplace distribution to each number in the histogram. We consider two aggregation techniques, SHE and THE.
  - **Summation with Histogram Encoding** (SHE) simply sums up the reported noisy histograms from all users.
  - **Thresholding with Histogram Encoding** (THE) is parameterized by a value θ; it interprets each noisy count above a threshold θ as a 1, and each count below θ as a 0.
- **Unary Encoding (UE).** An input v is encoded as a length-d bit vector, with only the bit corresponding to v set to 1. Here two key parameters in perturbation are p,

the probability that 1 remains 1 after perturbation, and q, the probability that 0 is perturbed into 1. Depending on their choices, we have two protocols, SUE and OUE.

  - **Symmetric Unary Encoding** (SUE) uses $p + q = 1$; this is the Basic Rappor protocol [2].
  - **Optimized Unary Encoding** (OUE) uses optimized choices of p and q; this is newly proposed In this section.

- **Local Hashing (LH).** An input v is encoded by choosing at random H from a universal hash function family $\mathbb{H}$, and then outputting $(H, H(v))$. This is called Local Hashing because each user chooses independently the hash function to use. Here a key parameter is the range of these hash functions. Depending on this range, we have two protocols, BLH and OLH.

  - **Binary Local Hashing** (BLH) uses hash functions that outputs a single bit. This is equivalent to the random matrix projection technique in [8].
  - **Optimized Local Hashing** (OLH) uses optimized choices for the range of hash functions; this is newly proposed In this section.

3.3.1 Direct Encoding (DE)

One natural method is to extend the binary response method to the case where the number of input values is more than 2.

Encoding and Perturbing. $\mathsf{Encode}_{\mathrm{DE}}(v) = v$, and Perturb is defined as follows.

$$\Pr\left[\mathsf{Perturb}_{\mathrm{DE}}(x) = i\right] = \begin{cases} p = \frac{e^\epsilon}{e^\epsilon + d - 1}, & \text{if } i = x \\ q = \frac{1-p}{d-1} = \frac{1}{e^\epsilon + d - 1}, & \text{if } i \neq x \end{cases} \tag{3.4}$$

Theorem 3.3.1 (Privacy of DE)**.** *The Direct Encoding (DE) Protocol satisfies ϵ-LDP.*

Proof: For any inputs v_1, v_2 and output y, we have:

$$\frac{\Pr\left[\mathsf{PE}_{\mathrm{DE}}(v_1) = y\right]}{\Pr\left[\mathsf{PE}_{\mathrm{DE}}(v_2) = y\right]} \leq \frac{p}{q} = \frac{e^\epsilon/(e^\epsilon + d - 1)}{1/(e^\epsilon + d - 1)} = e^\epsilon$$

□

Aggregation. Let the $\mathsf{Support}$ function for DE be $\mathsf{Support}_{\mathrm{DE}}(i) = \{i\}$, i.e., each output value i supports the input i. Then this protocol is pure, with $p^* = p$ and $q^* = q$.

Plugging these values into Equation (3.3), we have

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{DE}}(i)] = n \cdot \frac{d - 2 + e^\epsilon}{(e^\epsilon - 1)^2} \tag{3.5}$$

Note that the variance given above is linear in nd. As d increases, the accuracy of DE suffers. This is because, as d increases, $p = \frac{e^\epsilon}{e^\epsilon + d - 1}$, the probability that a value is transmitted correctly, becomes smaller. For example, when $e^\epsilon = 49$ and $d = 2^{16}$, we have $p = \frac{49}{65584} \approx 0.00075$.

3.3.2 Histogram Encoding (HE)

In Histogram Encoding (HE), an input $x \in [d]$ is encoded using a length-d histogram.

Encoding. $\mathsf{Encode}_{\mathrm{HE}}(v) = [0.0, 0.0, \cdots, 1.0, \cdots, 0.0]$, where only the v-th component is 1.0. Two different input v values will result in two vectors that have $\mathsf{L1}$ distance of 2.0.

Perturbing. $\mathsf{Perturb}_{\mathrm{HE}}(B)$ outputs B' such that $B'[i] = B[i] + \mathcal{L}\left(\frac{2}{\epsilon}\right)$,
where $\mathcal{L}(\beta)$ is the Laplace distribution where $\mathsf{Pr}[\mathcal{L}(\beta) = x] = \frac{1}{2\beta} e^{-|x|/\beta}$.

Theorem 3.3.2 (Privacy of HE)**.** *The Histogram Encoding protocol satisfies ϵ-LDP.*

Proof: For any inputs v_1, v_2, and output B, we have

$$\frac{\mathsf{Pr}[B|v_1]}{\mathsf{Pr}[B|v_2]} = \frac{\prod_{i\in[d]} \mathsf{Pr}[B[i]|v_1]}{\prod_{i\in[d]} \mathsf{Pr}[B[i]|v_2]} = \frac{\mathsf{Pr}[B[v_1]|v_1]\,\mathsf{Pr}[B[v_2]|v_1]}{\mathsf{Pr}[B[v_1]|v_2]\,\mathsf{Pr}[B[v_2]|v_2]} \leq e^{\epsilon/2} \cdot e^{\epsilon/2} = e^\epsilon$$

□

Aggregation: Summation with Histogram Encoding (SHE). works as follows: For each value, sum the noisy counts for that value reported by all users. That is, $\tilde{c}_{\mathrm{SHE}}(i) =$

$\sum_j B^j[i]$, where B^j is the noisy histogram received from user j. This aggregation method does not provide a Support function and is not pure. We prove its property as follows.

Theorem 3.3.3. *In* SHE*, the estimation* $\tilde{c}_{\text{SHE}}$ *is unbiased. Furthermore, the variance is*

$$\text{Var}^*[\tilde{c}_{\text{SHE}}(i)] = n\frac{8}{\epsilon^2}$$

Proof: Since the added noise is 0-mean; the expected value of the sum of all noisy counts is the true count.

The $\mathcal{L}(\beta)$ distribution has variance $\frac{2}{\beta^2}$, since $\beta = \frac{\epsilon}{2}$ for each $B^j[i]$, then the variance of each such variable is $\frac{8}{\epsilon^2}$, and the sum of n such independent variables have variance $n\frac{8}{\epsilon^2}$. □

Aggregation: Thresholding with Histogram Encoding (THE). interprets a vector of noisy counts discretely by defining

$$\text{Support}_{\text{THE}}(B) = \{v \mid B[v] > \theta\}$$

That is, each noise count that is $> \theta$ supports the corresponding value. This thresholding step can be performed either by the user or by the aggregator. It does not access the original value, and thus does not affect the privacy guarantee. Using thresholding to provide a Support function makes the protocol pure. The probability p^* and q^* are given by

$$p^* = 1 - F(\theta - 1); \;\; q^* = 1 - F(\theta),$$
$$\text{where } F(x) = \begin{cases} \frac{1}{2}e^{\frac{\epsilon}{2}x}, & \text{if } x < 0 \\ 1 - \frac{1}{2}e^{-\frac{\epsilon}{2}x}, & \text{if } x \geq 0 \end{cases}$$

Here, $F(\cdot)$ is the cumulative distribution function of Laplace distribution. If $0 \leq \theta \leq 1$, then we have

$$p^* = 1 - \frac{1}{2}e^{\frac{\epsilon}{2}(\theta-1)}; \;\; q^* = \frac{1}{2}e^{-\frac{\epsilon}{2}\theta}.$$

Plugging these values into Equation (3.3), we have

$$\mathrm{Var}^*[\tilde{c}_{\mathrm{HET}}(i)] = n \cdot \frac{2e^{\epsilon\theta/2} - 1}{(1 + e^{\epsilon(\theta-1/2)} - 2e^{\epsilon\theta/2})^2}$$

Comparing SHE **and** THE**.** Fixing ϵ, one can choose a θ value to minimize the variance. Numerical analysis shows that the optimal θ is in $(\frac{1}{2}, 1)$, and depends on ϵ. When ϵ is large, $\theta \to 1$. Furthermore, $\mathrm{Var}[\tilde{c}_{\mathsf{THE}}] < \mathrm{Var}[\tilde{c}_{\mathsf{SHE}}]$ is always true. This means that by thresholding, one improves upon directly summing up noisy counts, likely because thresholding limits the impact of noises of large magnitude. In Section 3.4.1, we illustrate the differences between them using actual numbers.

3.3.3 Unary Encoding (UE)

Basic Rappor, which we described in Section 3.1.1, takes the approach of directly perturbing a bit vector. We now explore this method further.

Encoding. $\mathsf{Encode}(v) = [0, \cdots, 0, 1, 0, \cdots, 0]$, a length-$d$ binary vector where only the v-th position is 1.

Perturbing. $\mathsf{Perturb}(B)$ outputs B' as follows:

$$\Pr\left[B'[i] = 1\right] = \begin{cases} p, & \text{if } B[i] = 1 \\ q, & \text{if } B[i] = 0 \end{cases}$$

Theorem 3.3.4 (Privacy of UE)**.** *The Unary Encoding protocol satisfies ϵ-LDP for*

$$\epsilon = \ln\left(\frac{p(1-q)}{(1-p)q}\right) \tag{3.6}$$

PROOF: For any inputs v_1, v_2, and output B, we have

$$\begin{aligned}\frac{\mathsf{Pr}\left[B|v_1\right]}{\mathsf{Pr}\left[B|v_2\right]} =& \frac{\prod_{i\in[d]}\mathsf{Pr}\left[B[i]|v_1\right]}{\prod_{i\in[d]}\mathsf{Pr}\left[B[i]|v_2\right]} && (3.7)\\ \leq& \frac{\mathsf{Pr}\left[B[v_1]=1|v_1\right]\mathsf{Pr}\left[B[v_2]=0|v_1\right]}{\mathsf{Pr}\left[B[v_1]=1|v_2\right]\mathsf{Pr}\left[B[v_2]=0|v_2\right]} && (3.8)\\ =& \frac{p}{q}\cdot\frac{1-q}{1-p} = e^{\epsilon}\end{aligned}$$

Equation (3.7) is because each bit is flipped independently, and Equation (3.8) is because v_1 and v_2 result in bit vectors that differ only in locations v_1 and v_2, and a vector with position v_1 being 1 and position v_2 being 0 maximizes the ratio. □

Aggregation. A reported bit vector is viewed as supporting an input i if $B[i] = 1$, i.e., $\mathsf{Support}_{\mathrm{UE}}(B) = \{i \mid B[i] = 1\}$. This yields $p^* = p$ and $q^* = q$. Interestingly, Equation (3.6) does not fully determine the values of p and q for a fixed ϵ. Plugging Equation (3.6) into Equation (3.3), we have

$$\begin{aligned}\mathrm{Var}^*[\tilde{c}_{\mathsf{UE}}(i)] &= \frac{nq(1-q)}{(p-q)^2} = \frac{nq(1-q)}{(\frac{e^{\epsilon}q}{1-q+e^{\epsilon}q}-q)^2}\\ &= n\cdot\frac{((e^{\epsilon}-1)q+1)^2}{(e^{\epsilon}-1)^2(1-q)q}. \qquad (3.9)\end{aligned}$$

Symmetric UE (SUE). Rappor's implementation chooses p and q such that $p + q = 1$; making the treatment of 1 and 0 symmetric. Combining this with Equation (3.6), we have

$$p = \frac{e^{\epsilon/2}}{e^{\epsilon/2}+1}, \quad q = \frac{1}{e^{\epsilon/2}+1}$$

Plugging these into Equation (3.9), we have

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{SUE}}(i)] = n\cdot\frac{e^{\epsilon/2}}{(e^{\epsilon/2}-1)^2}$$

Optimized UE (OUE). Instead of making p and q symmetric, we can choose them to minimize Equation (3.9). Take the partial derivative of Equation (3.9) with respect to q, and solving q to make the result 0, we get:

$$\begin{aligned}
\frac{\partial\left[\frac{((e^\epsilon-1)q+1)^2}{(e^\epsilon-1)^2(1-q)q}\right]}{\partial q} &= \frac{\partial\left[\frac{1}{(e^\epsilon-1)^2}\cdot\left(\frac{(e^\epsilon-1)^2 q}{1-q}+\frac{2(e^\epsilon-1)}{1-q}+\frac{1}{q(1-q)}\right)\right]}{\partial q} \\
&= \frac{\partial\left[\frac{1}{(e^\epsilon-1)^2}\cdot\left(-(e^\epsilon-1)^2+\frac{e^{2\epsilon}}{1-q}+\frac{1}{q}\right)\right]}{\partial q} \\
&= \frac{1}{(e^\epsilon-1)^2}\left(\frac{e^{2\epsilon}}{(1-q)^2}-\frac{1}{q^2}\right)=0 \\
\implies \frac{1-q}{q} &= e^\epsilon, \text{i.e.}, q=\frac{1}{e^\epsilon+1} \text{ and } p=\frac{1}{2}
\end{aligned}$$

Plugging $p=\frac{1}{2}$ and $q=\frac{1}{e^\epsilon+1}$ into Equation (3.9), we get

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{OUE}}(i)] = n\frac{4e^\epsilon}{(e^\epsilon-1)^2} \tag{3.10}$$

The reason why setting $p=\frac{1}{2}$ and $q=\frac{1}{e^\epsilon+1}$ is optimal when the true frequencies are small may be unclear at first glance; however, there is an intuition behind it. When the true frequencies are small, d is large. Recall that $e^\epsilon=\frac{p}{1-p}\frac{1-q}{q}$. Setting p and q can be viewed as splitting ϵ into $\epsilon_1+\epsilon_2$ such that $\frac{p}{1-p}=e^{\epsilon_1}$ and $\frac{1-q}{q}=e^{\epsilon_2}$. That is, ϵ_1 is the privacy budget for transmitting the 1 bit, and ϵ_2 is the privacy budget for transmitting each 0 bit. Since there are many 0 bits and a single 1 bit, it is better to allocate as much privacy budget for transmitting the 0 bits as possible. In the extreme, setting $\epsilon_1=0$ and $\epsilon_2=\epsilon$ means that setting $p=\frac{1}{2}$.

3.3.4 Binary Local Hashing (BLH)

Both HE and UE use unary encoding and have $\Theta(d)$ communication cost, which is too large for some applications. To reduce the communication cost, a natural idea is to first hash the input value into a domain of size $k<d$, and then apply the UE method to the hashed value. This is the basic idea underlying the Rappor method. However, a problem with this approach is that two values may be hashed to the same output, making them

indistinguishable from each other during decoding. Rappor tries to address this in several ways. One is to use more than one hash functions; this reduces the chance of a collision. The other is to use cohorts, so that different cohorts use different sets of hash functions. These remedies, however, do not fully eliminate the potential effect of collisions. Using more than one hash functions also means that every individual bit needs to be perturbed more to satisfy ϵ-LDP for the same ϵ.

A better approach is to make each user belong to a cohort by herself. We call this the **local hashing** approach. The random matrix-base protocol in [8] (described in Section 3.1.3), in its very essence, uses a local hashing encoding that maps an input value to a single bit, which is then transmitted using randomized response. Below is the Binary Local Hashing (BLH) protocol, which is logically equivalent to the one in Section 3.1.3, but is simpler and, we hope, better illustrates the essence of the idea.

Let $\mathbb{H}$ be a universal hash function family, such that each hash function $H \in \mathbb{H}$ hashes an input in $[d]$ into one bit. The universal property requires that

$$\forall x, y \in [d], x \neq y : \Pr_{H \in \mathbb{H}}[H(x) = H(y)] \leq \frac{1}{2}.$$

Encoding. $\mathsf{Encode}_{\mathrm{BLH}}(v) = \langle H, b \rangle$, where $H \leftarrow_R \mathbb{H}$ is chosen uniformly at random from $\mathbb{H}$, and $b = H(v)$. Note that the hash function H can be encoded using an index for the family $\mathbb{H}$ and takes only $O(\log n)$ bits.

Perturbing. $\mathsf{Perturb}_{\mathrm{BLH}}(\langle H, b \rangle) = \langle H, b' \rangle$ such that

$$\Pr[b' = 1] = \begin{cases} p = \frac{e^\epsilon}{e^\epsilon + 1}, & \text{if } b = 1 \\ q = \frac{1}{e^\epsilon + 1}, & \text{if } b = 0 \end{cases}$$

Aggregation. $\mathsf{Support}_{\mathsf{BLH}}(\langle H, b \rangle) = \{v \mid H(v) = b\}$, that is, each reported $\langle H, b \rangle$ supports all values that are hashed by H to b, which are half of the input values. Using this Support

function makes the protocol pure, with $p^* = p$ and $q^* = \frac{1}{2}p + \frac{1}{2}q = \frac{1}{2}$. Plugging the values of p^* and q^* into Equation (3.3), we have

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{BLH}}(i)] = n \cdot \frac{(e^\epsilon + 1)^2}{(e^\epsilon - 1)^2}.$$

3.3.5 Optimal Local Hashing (OLH)

Once the random matrix projection protocol is cast as binary local hashing, we can clearly see that the encoding step loses information because the output is just one bit. Even if that bit is transmitted correctly, we can get only one bit of information about the input, i.e., to which half of the input domain does the value belong. When ϵ is large, the amount of information loss in the encoding step dominates that of the random response step. Based on this insight, we generalize Binary Local Hashing so that each input value is hashed into a value in $[g]$, where $g \geq 2$. A larger g value means that more information is being preserved in the encoding step. This is done, however, at a cost of more information loss in the random response step. As in our analysis of the Direct Encoding method, a large domain results in more information loss.

Let $\mathbb{H}$ be a universal hash function family such that each $H \in \mathbb{H}$ outputs a value in $[d']$.

Encoding. $\mathsf{Encode}(v) = \langle H, x \rangle$, where $H \in \mathbb{H}$ is chosen uniformly at random, and $x = H(v)$.

Perturbing. $\mathsf{Perturb}(\langle H, x \rangle) = (\langle H, y \rangle)$, where

$$\forall_{i \in [d']} \, \mathsf{Pr}\left[y = i\right] = \begin{cases} p = \frac{e^\epsilon}{e^\epsilon + d' - 1}, & \text{if } x = i \\ q = \frac{1}{e^\epsilon + d' - 1}, & \text{if } x \neq i \end{cases}$$

Theorem 3.3.5 (Privacy of LH)**.** *The Local Hashing (*LH*) Protocol satisfies ϵ-LDP*

Proof: For any two possible input values v_1, v_2 and any output $\langle H, y \rangle$, we have,

$$\frac{\mathsf{Pr}\left[\langle H, y \rangle | v_1\right]}{\mathsf{Pr}\left[\langle H, y \rangle | v_2\right]} = \frac{\mathsf{Pr}\left[\mathsf{Perturb}(H(v_1)) = y\right]}{\mathsf{Pr}\left[\mathsf{Perturb}(H(v_2)) = y\right]} \leq \frac{p}{q} = e^\epsilon$$

□

Aggregation. Let $\mathsf{Support}_{\mathsf{LH}}(\langle H, y\rangle) = \{i \mid H(i) = y\}$, i.e., the set of values that are hashed into the reported value. This gives rise to a pure protocol with

$$p^* = p \text{ and } q^* = \frac{1}{d'}p + \frac{d'-1}{d'}q = \frac{1}{d'}.$$

Plugging these values into Equation (3.3), we have the

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{LP}}(i)] = n \cdot \frac{(e^\epsilon - 1 + d')^2}{(e^\epsilon - 1)^2(d'-1)}. \tag{3.11}$$

Optimized LH (OLH). Now we find the optimal d' value, by taking the partial derivative of Equation (3.11) with respect to d'.

$$\begin{aligned}\frac{\partial\left[\frac{(e^\epsilon-1+d')^2}{(e^\epsilon-1)^2(d'-1)}\right]}{\partial d'} &= \frac{\partial\left[\frac{d'-1}{(e^\epsilon-1)^2} + \frac{1}{d'-1}\cdot\frac{e^{2\epsilon}}{(e^\epsilon-1)^2} + \frac{2e^\epsilon}{(e^\epsilon-1)^2}\right]}{\partial d'} \\ &= \frac{1}{(e^\epsilon-1)^2} - \frac{1}{(d'-1)^2}\cdot\frac{e^{2\epsilon}}{(e^\epsilon-1)^2} = 0 \\ &\implies d' = e^\epsilon + 1\end{aligned}$$

When $d' = e^\epsilon + 1$, we have $p^* = \frac{e^\epsilon}{e^\epsilon+d'-1} = \frac{1}{2}$, $q^* = \frac{1}{d'} = \frac{1}{e^\epsilon+1}$ into Equation (3.9), and

$$\mathrm{Var}^*[\tilde{c}_{\mathsf{OLH}}(i)] = n \cdot \frac{4e^\epsilon}{(e^\epsilon-1)^2}. \tag{3.12}$$

Comparing OLH with OUE. It is interesting to observe that the variance we derived for optimized local hashing (OLH), i.e., Equation (3.12) is exactly that we have for optimized unary encoding (OUE), i.e., Equation (3.10). Furthermore, the probability values p^* and q^* are also exactly the same. This illustrates that OLH and OUE are in fact deeply connected. OLH can be viewed as a compact way of implementing OUE. Compared with OUE, OLH has communication cost $O(\log n)$ instead of $O(d)$.

The fact that optimizing two apparently different encoding approaches, namely, unary encoding and local hashing, results in conceptually equivalent protocol, seems to suggest that this may be optimal (at least when d is large). However, whether this is the best possible protocol remains an interesting open question.

3.4 Discussion

3.4.1 Which Protocol to Use

We have cast most of the LDP protocols proposed in the literature into our framework of pure LDP protocols. Doing so also enables us to generalize and optimize existing protocols. Now we are able to answer the question: Which LDP protocol should one use in a given setting?

Guideline. Table 3.1 lists the major parameters for the different protocols. Histogram encoding and unary encoding requires $\Theta(d)$ communication cost, and is expensive when d is large. Direct encoding and local hashing require $\Theta(\log d)$ or $\Theta(\log n)$ communication cost, which amounts to a constant in practice. All protocols other than DE have $O(n \cdot d)$ computation cost to estimate frequency of all values.

Numerical values of the approximate variances using Equation (3.3) for all protocols are given in Table 3.2 and Figure 3.1 ($n = 10,000$). Our analysis gives the following guidelines for choosing protocols.

- When d is small, more precisely, when $d < 3e^{\epsilon}+2$, DE is the best among all approaches.
- When $d > 3e^{\epsilon}+2$, and the communication cost $\Theta(d)$ is acceptable, one should use OUE. (OUE has the same variance as OLH, but is easier to implement and faster because no hash functions is used.)
- When d is so large that the communication cost $\Theta(d)$ is too large, we should use OLH. It offers the same accuracy as OUE, but has communication cost $O(\log d)$ instead of $O(d)$.

Discussion. In addition to the guidelines, we make the following observations. Adding Laplacian noises to a histogram is typically used in a setting with a trusted data curator,

Table 3.1. Comparison of communication cost and variances for different methods.

	DE	SHE	THE ($\theta = 1$)	SUE	OUE	BLH	OLH
Communication Cost	$O(\log d)$	$O(d)$	$O(d)$	$O(d)$	$O(d)$	$O(\log n)$	$O(\log n)$
$\mathrm{Var}[\tilde{c}(i)]/n$	$\frac{d-2+e^\epsilon}{(e^\epsilon-1)^2}$	$\frac{8}{\epsilon^2}$	$\frac{2e^{\epsilon/2}-1}{(e^{\epsilon/2}-1)^2}$	$\frac{e^{\epsilon/2}}{(e^{\epsilon/2}-1)^2}$	$\frac{4e^\epsilon}{(e^\epsilon-1)^2}$	$\frac{(e^\epsilon+1)^2}{(e^\epsilon-1)^2}$	$\frac{4e^\epsilon}{(e^\epsilon-1)^2}$

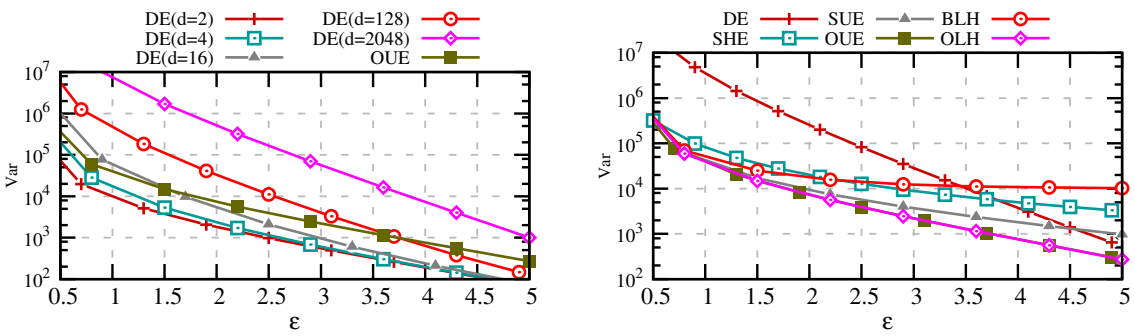
Table 3.2. Numerical values of $\mathrm{Var}[\tilde{c}(i)]/n$ for different methods.

	DE ($d = 2$)	DE ($d = 32$)	DE ($d = 2^{10}$)	SHE	THE ($\theta = 1$)	SUE	OUE	BLH	OLH
$\epsilon = 0.5$	3.92	75.20	2432.40	32.00	19.44	15.92	15.67	16.67	15.67
$\epsilon = 1.0$	0.92	11.08	347.07	8.00	5.46	3.92	3.68	4.68	3.68
$\epsilon = 2.0$	0.18	0.92	25.22	2.00	1.50	0.92	0.72	1.72	0.72
$\epsilon = 4.0$	0.02	0.03	0.37	0.50	0.34	0.18	0.08	1.08	0.08

who first computes the histogram from all users' data and then adds the noise. SHE applies it to each user's data. Intuitively, this should perform poorly relative to other protocols specifically designed for the local setting. However, SHE performs very similarly to BLH, which was specifically designed for the local setting. In fact, when $\epsilon > 2.5$, SHE performs better than BLH.

While all protocols' variances depend on ϵ, the relationships are different. BLH is least sensitive to change in ϵ because binary hashing loses too much information. Indeed, while all other protocols have variance goes to 0 when ϵ goes to infinity, BLH has variance goes to n. SHE is slightly more sensitive to change in ϵ. DE is most sensitive to change in ϵ; however, when d is large, its variance is very high. OLH and OUE are able to better benefit from an increase in ϵ, without suffering the poor performance for small ϵ values.

Another interesting finding is that when $d = 2$, the variance of DE is $\frac{e^\epsilon}{(e^\epsilon-1)^2}$, which is exactly $\frac{1}{4}$ of that of OUE and OLH, whose variances do not depend on d. Intuitively, it is easier to transmit a piece of information when it is binary, i.e., $d = 2$. As d increases, one needs to "pay" for this increase in source entropy by having higher variance. However, it seems that there is a cap on the "price" one must pay no matter how large d is, i.e., OLH's variance does not depend on d and is always 4 times that of DE with $d = 2$. There may exist a deeper reason for this rooted in information theory. Exploring these questions is beyond the scope of this paper.



(a) Vary ϵ (b) Vary ϵ (fixing $d = 2^{10}$)

Figure 3.1. Numerical values of $\mathrm{Var}[\tilde{c}(i)]$ for different methods.

3.4.2 On Answering Multiple Questions

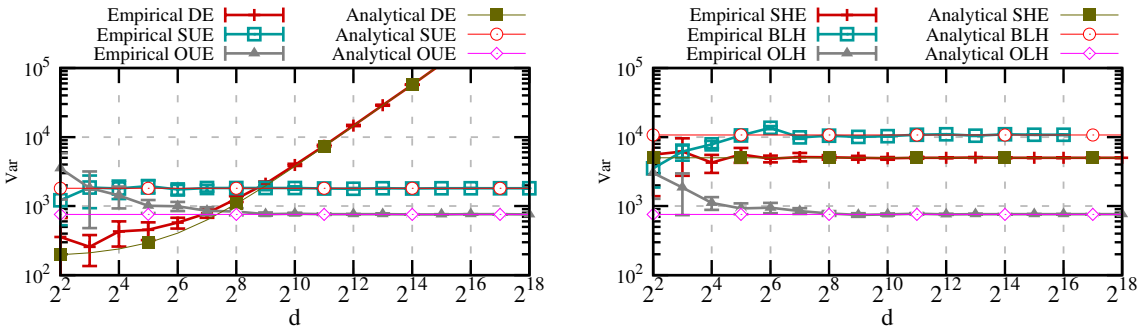
In the setting of traditional DP, the privacy budget is split when answering multiple queries. In the local setting, previous work follow this tradition and let the users split privacy budget evenly and report on multiple questions. Instead, we suggest partitioning the users randomly into groups, and letting each group of users answer a separate question. In fact, this is an important principle of LDP that we keep using in the next chapters. Now we compare the utilities by these approaches.

Suppose there are $Q \geq 2$ questions. We calculate variances on one question. Since there are different number of users in the two cases (n versus n/Q), we normalize the estimations into the range from 0 to 1. In OLH, the variance is $\sigma^2 = \mathrm{Var}^*[\tilde{c}_{\mathsf{OLH}}(i)/n] = \frac{4e^\epsilon}{(e^\epsilon-1)^2 \cdot n}$.

When partitioning the users, n/Q users answer one question, rendering $\sigma_1^2 = \frac{4Qe^\epsilon}{(e^\epsilon-1)^2 \cdot n}$; when privacy budget is split, ϵ/Q is used for one question, we have $\sigma_2^2 = \frac{4e^{\epsilon/Q}}{\left(e^{\epsilon/Q}-1\right)^2 \cdot n}$. We want to show $\sigma_1^2 < \sigma_2^2$:

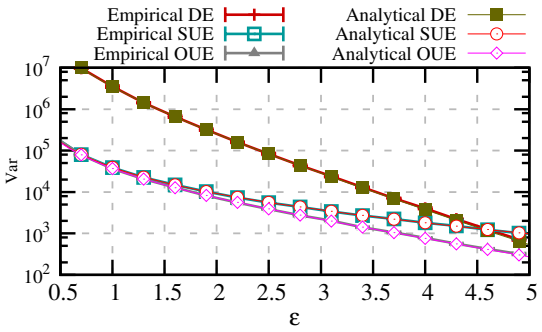
$$\sigma_2^2 - \sigma_1^2 = \frac{4}{n}\left(\frac{e^{\epsilon/Q}}{\left(e^{\epsilon/Q}-1\right)^2} - \frac{Qe^\epsilon}{\left(e^\epsilon-1\right)^2}\right)$$

$$= \frac{4e^{\epsilon/Q}}{n\left(e^{\epsilon/Q}-1\right)^2\left(e^\epsilon-1\right)^2} \cdot \left[\left(e^\epsilon-1\right)^2 - Qe^{\epsilon-\epsilon/Q}\left(e^{\epsilon/Q}-1\right)^2\right]$$

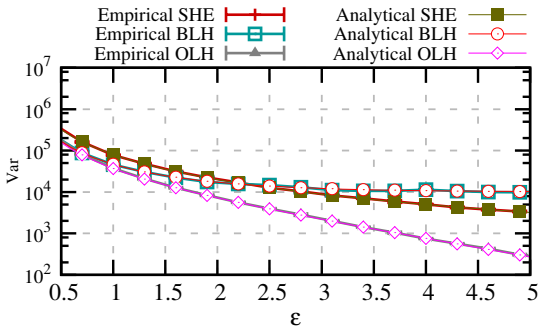


(a) Vary d (fixing $\epsilon = 4$)

(b) Vary d (fixing $\epsilon = 4$)



(c) Vary ϵ (fixing $d = 2^{10}$)



(d) Vary ϵ (fixing $d = 2^{10}$)

Figure 3.2. Comparing empirical and analytical variance.

The first term is always greater than zero since $\epsilon > 0$. For the second term, we define $e^{\epsilon/Q} = z$, and write it as:

$$(z^Q - 1)^2 - Qz^{Q-1}(z-1)^2 = (z-1)^2 \cdot \left[(z^{Q-1} + z^{Q-2} + \ldots + 1)^2 - Qz^{Q-1}\right] > 0$$

Therefore, σ_1^2 is always smaller than σ_2^2. Thus utility of partitioning users is better than splitting privacy budget.

3.5 Experimental Evaluation

We empirically evaluate these protocols on both synthetic and real-world datasets. All experiments are performed ten times and we plot the mean and standard deviation.

3.5.1 Verifying Correctness of Analysis

The conclusions we drew above are based on analytical variances. We now show that our analytical results of variances match the empirically measured squared errors. For the empirical data, we issue queries using the protocols and measure the average of the squared errors, namely, $\frac{1}{d}\sum_{i\in[d]}\left[\tilde{c}(i) - nf_i\right]^2$, where f_i is the fraction of users taking value i. We run queries for all i values and repeat for ten times. We then plot the average and standard deviation of the squared error. We use synthetic data generated by following the Zipf's distribution (with distribution parameter $s = 1.1$ and $n = 10,000$ users), similar to experiments in [2].

Figure 3.2 gives the empirical and analytical results for all methods. In Figures 3.2a and 3.2b, we fix $\epsilon = 4$ and vary the domain size. For sufficiently large d (e.g., $d \geq 2^6$), the empirical results match very well with the analytical results. When $d < 2^6$, the analytical variance tends to underestimate the variance, because in Equation (3.3) we ignore the f_i terms. Standard deviation of the measured squared error from different runs also decreases when the domain size increases. In Figures 3.2c and 3.2d, we fix the domain size to $d = 2^{10}$ and vary the privacy budget. We can see that the analytical results match the empirical results for all ϵ values and all methods.

In practice, since the group size d' of OLH can only be integers, we round $d' = e^\epsilon + 1$ to the nearest integer.

3.5.2 Towards Real-world Estimation

We run OLH, BLH, together with Rappor, on real datasets. The goal is to understand how does each protocol perform in real world scenarios and how to interpret the result. Note that Rappor does not fall into the pure framework of LDP protocols so we cannot use Theorem 3.2.2 to obtain the variance analytically. Instead, we run experiments to examine its performance empirically. Following the setting of Erlingsson et al. [2], we use a 128-bit Bloom filter, 2 hash functions and 8/16 cohorts in Rappor. In order to vary ϵ, we tweak the f value. The instantaneous randomization process is omitted. We implement Rappor in

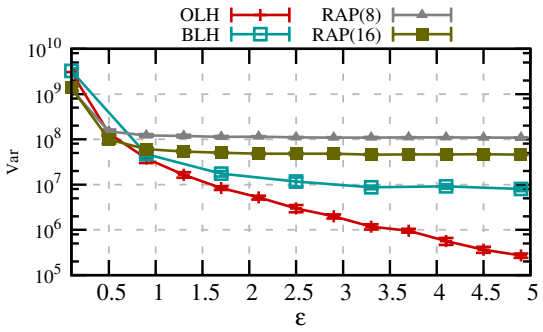


Figure 3.3. Average squared error, varying ϵ.

Python. The regression part, which Rappor introduces to handle the collisions in the Bloom filter, is implemented using Scikit-learn library [9].

Datasets. We use the **Kosarak** dataset [10], which contains the click stream of a Hungarian news website. There are around 8 million click events for 41,270 different pages. The goal is to estimate the popularity of each page, assuming all events are reported.

Accuracy on Frequent Values

One goal of estimating a distribution is to find out the frequent values and accurately estimate them. We run different methods to estimate the distribution of the Kosarak dataset. After the estimation, we issue queries for the 30 most frequent values in the original dataset. We then calculate the average squared error of the 30 estimations produced by different methods. Figure 3.3 shows the result. We try Rappor with both 8 cohorts (RAP(8)) and 16 cohorts (RAP(16)). It can be seen that when $\epsilon > 1$, OLH starts to show its advantage. Moreover, variance of OLH decreases fastest among the four. Due to the internal collision caused by Bloom filters, the accuracy of Rappor does not benefit from larger ϵ. We also perform this experiment on different datasets, and the results are similar.

Distinguish True Counts from Noise

Although there are noises, infrequent values are still unlikely to be estimated to be frequent. Statistically, the frequent estimates are more reliable, because the probability it

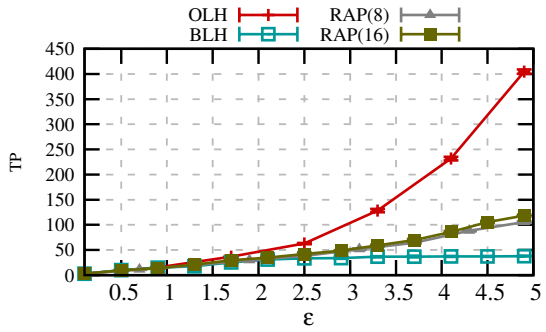
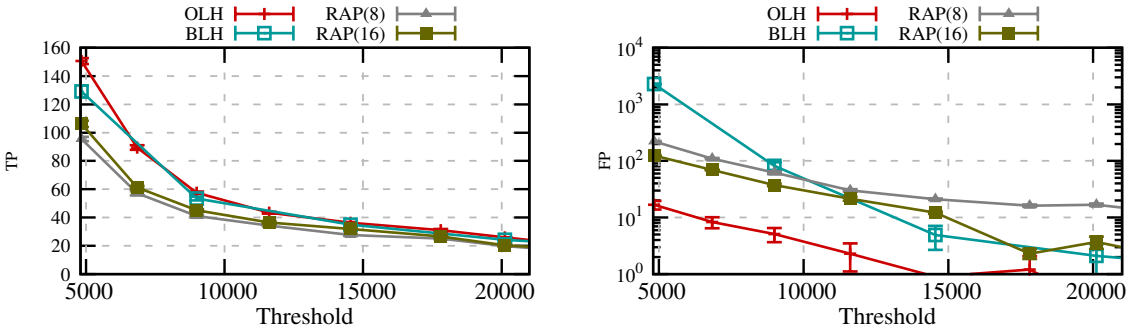


Figure 3.4. Number of true positives, varying ϵ, using significance threshold. The dashed line corresponds to the average number of items identified.



(a) Number of True Positives

(b) Number of False Positives

Figure 3.5. Results on Kosarak dataset. The y axes are the number of identified hash values that is true/false positive. The x axes are the threshold. We assume $\epsilon = 4$.

is generated from an infrequent value is quite low. However, for the infrequent estimates, we do not know whether it comes from an originally infrequent value or a zero-count value. Therefore, after getting the estimation, we need to choose which estimate to use, and which to discard.

Significance Threshold. In [2], the authors propose to use the significance threshold. After the estimation, all estimations above the threshold are kept, and those below the threshold T_s are discarded.

$$T_s = \Phi^{-1}\left(1 - \frac{\alpha}{d}\right)\sqrt{\mathrm{Var}^*} \tag{3.13}$$

where d is the domain size, Φ^{-1} is the inverse of the cumulative density function of standard normal distribution, and the term inside the square root is the variance of the protocol. Roughly speaking, the parameter α controls the number of values that originally have low frequencies but estimated to have frequencies above the threshold (also known as false positives). We use $\alpha = 0.05$ in our experiment.

For the values whose estimations are discarded, we do not know for sure whether they have low or zero frequencies. Thus, a common approach is to assign the remaining probability to each of them uniformly.

Recall Var* is the term we are trying to minimize. So a protocol with a smaller variance will have a lower threshold; thus more values can be detected reliably.

Number of Reliable Estimation. We run different protocols using the significance threshold T_s on the Kosarak dataset. Note that T_s will change as ϵ changes. We define a true (false) positive as a value that has frequency above (below) the threshold, and is estimated to have frequency above the threshold. In Figure 3.4, we show the number of true positives versus ϵ. As ϵ increases, the number of true positives increases. When $\epsilon = 4$, Rappor can output 75 true positives, BLH can only output 36 true positives, but OLH can output nearly 200 true positives. We also notice that the output sizes are similar for Rappor and OLH, which indicates that OLH gives out very few false positives compared to Rappor. The cohort size does not affect much in this setting.

On Information Quality

Now we test both the number of true positives and false positives, varying the threshold. We run OLH, BLH and Rappor on the Kosarak dataset.

As we can see in Figure 3.5a, fixing a threshold, OLH and BLH performs similarly in identifying true positives, which is as expected, because frequent values are rare, and variance does not change much the probability it is identified. Rappor performs slightly worse because of the Bloom filter collision.

As for the false positives, as shown in Figure 3.5b, different protocols perform quite differently in eliminating false positives. When fixing T_s to be $5,000$, OLH produces tens

of false positives, but BLH will produce thousands of false positives. The reason behind this is that, for the majority of infrequent values, their estimations are directly related to the variance of the protocol. A protocol with a high variance means that more infrequent values will become frequent during estimation. As a result, because of its smallest Var^*, OLH produces the least false positives while generating the most true positives.

4. HEAVY HITTER IDENTIFICATION

(A version of this chapter has been previously published in IEEE TDSC 2019 [11].

There is also a public version on arXiv with some additional information [12].)

In the heavy hitter identification problem, the aggregator wants to know the distribution of the frequent values. When the data domain D is relatively small, having a frequency estimation protocol (or frequency oracle, as described in Chapter 3) suffices, as the aggregator can invoke the frequency oracle for all values in D, and identify the frequent ones. However, in many applications, the data domain D is very large, e.g., 2^{128} when the input values have 16 bytes. Enumerating through all values in them is computationally infeasible.

In this chapter we focus on the problem of identifying frequent values under the local differential privacy (LDP) setting when the input domain is large. For simplicity, we assume that each value is represented by a binary string of length m, although our method can be easily changed to support more complicated structure of values, such as a value consisting of multiple components.

More formally, the problem of finding frequent values (heavy hitters) can be defined either as identifying the top-k values or finding values that appear above a certain threshold. We assume that each user has a single value, and thus each frequency threshold can be approximately translated into a k value. We use the top-k version of definition: Given n values $v^1, v^2, \ldots, v^n$ from domain D, an element $v \in D$ is a top-k heavy hitter if its frequency $f_v = \frac{|\{j|j\in[n]\wedge v^j=v\}|}{n}$ is ranked among top k frequencies of all possible values.

Suppose that each user has a length $m = 128$ binary string v as input value, the naive approach of querying the frequency of each string requires 2^{128} oracle queries and is infeasible.

The Optimized Local Hashing (OLH) protocol presented earlier deals with a large domain size. But each invocation of the frequency oracle takes time linear in the population size. Furthermore, the computations needed for recovering the frequency of one value are independent from those needed for recovering that of another value.

The importance of population size. Note the the variances of GRR and OLH (Equations 3.5 and 3.12) are both linear in n. This is also true with all existing frequency oracles

under LDP, and is a fundamental accuracy cost one has to pay in order to achieve LDP [13]. This means that LDP protocols can be useful only when the group size n is large, and LDP protocols are meaningful only for the frequent values.

This also means that the standard deviation (SD) of the estimations is linear in $\sqrt{n}$. When n increases, the standard deviation of the absolute count increases; however, we are mostly interested in the normalized frequency (estimated frequency divided by the population size). The variance on estimated normalized frequency thus has SD that is linear in $\frac{1}{\sqrt{n}}$. A larger population size will thus lead to more accurate estimations.

4.1 Existing Solutions

4.1.1 The Segment Pairs Method (SPM)

The approach taken by Google's team lets each user report a pair of two randomly chosen segments [14]. We call this the Segment Pair Method (SPM).

In SPM, a length-m value is divided into g segments of length $s = m/g$ each. For example, when $g = 8, m = 128$, each segment has $s = 16$ bits. We use the notation $v[i : j]$ to denote the segment of v starting at the i'th bit and stopping just before the j'th bit. Thus $v[0 : m]$ represents the complete v. In addition to reporting the overall value v, a user also randomly chooses two segments to report. More specifically, the user randomly chooses $1 \leq \alpha \neq \beta \leq g$, and reports

$$\langle \mathcal{A}(v), \alpha, \beta, \mathcal{A}(v[(\alpha - 1)s : \alpha s]), \mathcal{A}(v[(\beta - 1)s : \beta s]) \rangle.$$

where $\mathcal{A}(\cdot)$ denotes the encode-perturbation algorithm of any LDP protocol described in Chapter 3. The user runs three reporting protocols in parallel, each using one third of privacy budget. Since each user randomly chooses 2 out of g segments to report, the population is divided into $\binom{g}{2}$ groups, each reporting for one pair of segments. When n users are reporting, one expects that about $\frac{n}{g/2}$ users report on each segment, and about $\frac{n}{g(g-1)/2}$ users report each pair of segments.

The aggregator first identifies the frequent patterns in each of the g segments, denoted by $C_1, \ldots, C_g$. Then, it queries, for each pair $1 \leq i, j \leq g$ of segments, the frequency for the values in $C_i \times C_j$, where Cartesian product operation $\times$ is defined as $C_i \times C_j = \{c_i||c_j : c_i \in C_i, c_j \in C_j\}$, and $||$ is the string concatenation operation, and identifies the value pairs that are frequent in segments i, j.

From the frequent value pairs for each pair of segments, the aggregator recovers candidates for frequent values for the whole domain, using the a priori principle that if a value $v \in D$ is frequent, every pair of its segments must also be frequent. Because of this filtering by segment pairs, the size of C is typically small enough to query the frequency of each value in it.

The main limitation of this method is that, since the length of each segment must be relatively small (one needs to enumerate through all possible values for each segment), when the domain is large, there are too many pairs of segments. As a result, the number of users reporting on each location-pair is limited, making it difficult to accurately identify frequent value pairs. For example, when $g = 16$, each pair has only about $\frac{n}{120}$ users.

4.1.2 The Multiple Channel Method (HASH)

Bassily and Smith proposed an approach which we call Multiple Channel Method (HASH) [8]. Our description of HASH below simplifies that in [8], and is equivalent to it. This approach separates the values into multiple channels so that with high probability each channel has at most one frequent value, and then identifies this candidate frequent value by identifying each bit of it.

The approach uses a hash function H that maps each input value v to an integer in $\{1 \ldots h\}$. We say that v is mapped to the channel $H(v)$. The value h needs to be large enough to ensure that the probability that any two frequent values are mapped to the same channel is low. Each user with input v randomly selects i such that $0 \leq i < m$ and reports:

$$\langle \mathcal{A}(v), i, b_1, b_2, \cdots, b_h \rangle$$

The privacy budget ϵ is divided into two parts $\epsilon_1 + \epsilon_2 = \epsilon$. Sending the value v uses ϵ_1; b_j's are computed such that when $j \neq H(v)$, b is a randomly sampled bit, and when $j = H(v)$, b is a perturbed value of the $v[i]$, flipped with probability $q = \frac{1}{e^{\epsilon_2}+1}$. That is, each user chooses one of the m bit to report.

From each channel, the aggregator extracts a candidate frequent value by taking the majority vote for each bit. The aggregator then queries the frequency of these candidates and outputs the frequent values.

One main limitation of this approach is that since each user reports a single bit, only a small number of users are reporting for each bit. For example, with $m = 128$, only $\frac{n}{128}$ users participate in the determination of candidate for each bit. Furthermore, to correctly recover the candidate value, each of the 128 bits must be recovered correctly. (While error correction code is suggested in [8], that will further reduce the group size and increase the probability that any one bit is recovered correctly.) This limitation can be addressed by having each user report a bigger block (such as 16 bit) at a time, which does improve the accuracy.

Another limitation is that since one identifies a single candidate from each channel, each user has to report on multiple channels, and the oracle queries must be made on all h channels. This adds a multiplicative factor of h to the communication and computation overheads.

4.2 The Prefix Extending Method

In both SPM and HASH, to deal with the challenge of large domains, a bit string input is divided into *non-overlapping* segments so that one can recover frequent patterns in each segment. These patterns need to be combined into a set of candidate frequent values. SPM does this by making each user report a pair of segments, dividing the population into $\binom{g}{2}$ groups. HASH does this by using multiple channels so that within each channel one focuses on identifying a single candidate frequent value.

We observe that instead of dividing a bit string into non-overlapping segments, one can use segments that overlap with each other. In particular, one can consider prefixes of different

lengths. We propose the Prefix Extending Method (PEM), which gradually identifies longer and longer frequent prefixes.

4.2.1 Overview of Prefix Extending Method (PEM)

The PEM method is parameterized by the parameter g. The population is divided into g mutually exclusive groups. A user is randomly assigned into one of g groups. (The assignment of users to groups can be made by the aggregator, or by having each user selecting a group at random.) The groups are indexed from 1 to g. To identify k heavy hitters, users in the i-th group use an FO protocol to report the prefix of length

$$s_i = \lceil \log_2 k \rceil + \left\lceil i \times \frac{m - \lceil \log_2 k \rceil}{g} \right\rceil.$$

Let $D_1 = \{0,1\}^{s_1}$, the aggregator uses the first group's reports to identify which values in D_1 are frequent prefixes. Let C_1 be the result. The aggregator then constructs $D_2 = C_1 \times \{0,1\}^{s_2 - s_1}$, which are candidates for longer frequent prefixes, and uses the second group's reports to identify the frequent ones as C_2. This continues until the last step where C_g gives the set of frequent values.

Example 1. *For example, consider the case that D consists of length-128 binary strings, and we want to identify the $2^8 = 256$ most frequent values. We could divide users into $g = 6$ groups, with the i'th group ($1 \le i \le 6$) using an FO protocol to report the prefix of length $8 + 20i$. That is, the first group reports 28-bit prefixes, the second group reports 48-bit prefixes, and so on. The 6th and last group reports the 128-bit values. Using reports from the first group, and querying the frequency of every one of the 2^{28} length-28 prefix, we can discover C_1, the 2^8 most frequent length-28 prefixes. Using reports from the second group, and querying the 2^{28} strings obtained by appending a string in C_1 with each length-20 binary string, we can discover C_2, the 2^8 most frequent length-48 prefixes. By iterating the above step, we can discover the 2^8 most frequent values with a total of 6×2^{28} FO queries.*

Algorithm 1 gives the outline of PEM. It is executed between the server and the users. The server iterates through the groups. In group i for $i = 1, 2, \ldots, g$, the server constructs

Algorithm 1 PEM

Input: n, m, k, ϵ, k, g.
Output: Top-k heavy hitters C_g and their frequencies.

procedure PEM(n, m, k, ϵ)
  Partition users into g groups
  Init $s_0 \leftarrow \lceil \log_2 k \rceil, C_0 \leftarrow \{0,1\}^{s_0}$
  for $i = 1$ to g **do** ▷ Server side
    $s_i \leftarrow \lceil \log_2 k \rceil + \lceil i \times \frac{m - \lceil \log_2 k \rceil}{g} \rceil$
    Construct $D_i = C_{i-1} \times \{0,1\}^{s_i - s_{i-1}}$
    Receive reports from users
    Estimate frequency of all $v \in D_i$
    Construct C_i as k highest estimated values
  for $j = 1$ to n **do** ▷ User side
    User j gets group assignment i
    $s_i \leftarrow \lceil \log_2 k \rceil + \lceil i \times \frac{m - \lceil \log_2 k \rceil}{g} \rceil$
    User j reports $\langle i, \mathcal{A}(v^j[0 : s_i]) \rangle$

the domain D_i, receives the reports from users in that group, and estimates the values in D_i using the frequency oracle. The top-k highly estimated values are identified as the prefix candidates $C_i \subset D_i$. The final results are C_g and its corresponding estimations. On the user side, each user just reports a prefix of his private value using the frequency oracle.

Privacy Guarantee. We show that when using any FO protocol that satisfies ϵ-LDP, PEM satisfies ϵ-LDP.

Theorem 4.2.1 (Privacy of PEM)**.** PEM *satisfies ϵ-LDP when it uses an* FO *protocol that satisfies ϵ-LDP.*

PROOF: For each user, the report of PEM consists of an index i, which is determined indepedent of the user's private input, and an output from the underlying frequency oracle. For any input $v_1, v_2 \in D$, and for any specific tuple $\langle i, y\rangle \in Range(\mathsf{PEM})$, we have

$$\begin{aligned}
&\max_{v_1,v_2,\langle i,y\rangle} \frac{\Pr\left[\mathsf{PEM}(v_1) = \langle i, y\rangle\right]}{\Pr\left[\mathsf{PEM}(v_2) = \langle i, y\rangle\right]} \\
=&\max_{v_1,v_2,\langle i,y\rangle} \frac{\Pr\left[\mathcal{A}(v_1[0:s_i]) = y|i\right] \cdot \Pr\left[i\right]}{\Pr\left[\mathcal{A}(v_2[0:s_i]) = y|i\right] \cdot \Pr\left[i\right]} \qquad (4.1)\\
=&\max_{v_1,v_2,y} \frac{\Pr\left[\mathcal{A}(v_1[0:s_i]) = y\right]}{\Pr\left[\mathcal{A}(v_2[0:s_i]) = y\right]} \\
\leq& e^{\epsilon} \qquad (4.2)
\end{aligned}$$

$\Pr[i]$ from line (4.1) represents the probability the user chooses i. Since it is random and independent of the private value, it is $1/g$. The last line (4.2) is derived from the theorem that the FO perturb function $\mathcal{A}$ is ϵ-LDP. □

4.2.2 Instantiation and Analysis of PEM

Dividing population instead of budget. Note that we divide users into groups and require each user to answer a separate question. An alternative is to let each user report $v[0:s_i]$ for all $i \in 1, \ldots, g$, and use ϵ/g as the privacy parameter for each reporting. This approach, however, will deteriorates utility, as pointed out in Section 3.4.2.

Choice of FO. One can use different FO protocols to instantiate PEM. So long as the FO protocol satisfies ϵ-LDP, privacy is satisfied. The criteria is accuracy, communication, and computation cost. In the setting of large domains, we propose to use OLH, since it achieves optimal accuracy and communication cost. More importantly, it makes the overall protocol non-interactive: each user hashes his prefix, and the aggregator only estimates values that are interested. To estimate the frequency of one value, the server needs to evaluate n/g hash function, where n/g is the number of user reports used for the estimation.

Complexity analysis. The server-side computation is dominated by the number of queries to the frequency oracle. Specifically, constructing each C_i requires evaluating the frequency

of $|D_i| = k \cdot 2^{s_i - s_{i-1}} \simeq k \cdot 2^{\lceil \frac{m - \lceil \log_2 k \rceil}{g} \rceil}$ values. Note that one estimation requires evaluating n/g hash functions, and there are g groups. Therefore, the total number of oracle queries is

$$k \cdot 2^{\lceil \frac{m - \lceil \log_2 k \rceil}{g} \rceil} \cdot \frac{n}{g} \cdot g = nk \cdot 2^{\lceil \frac{m - \lceil \log_2 k \rceil}{g} \rceil}. \tag{4.3}$$

Observe that the computation complexity grows with smaller g. Also note that within each iteration, the evaluation for each hash function can be paralleled.

Example 2. *Continuing Example 1, instead of dividing the users into* 6 *groups, we could divide the users into* $g = 12$ *groups (with each successive group reporting prefixes that are* 10 *bit longer). This would require* 12×2^{20} FO *queries in total. We could also divide the users into* $g = 4$ *groups, which requires* 4×2^{38} FO *queries in total.*

4.2.3 Concurrent work: PEM1.

After we finished this work (and made public [12]), we found a simultaneous paper [15] by Bassily et al. This paper proposes two methods to handle the heavy hitter problem. The first method is similar to our PEM protocol except that each group of users report on one incremental bit. We call it PEM1. This divides users into m groups. Since this line of work is motivated by the applications where m is large, dividing the population into m groups will result in poor accuracy. We include comparision with this method in our experiments.

The other method is basically the HASH method with only $\sqrt{n}$ channels (instead of $n^{1.5}$ channels, as suggested in [8]). In our experimental comparison with HASH in [12], we already used around $\sqrt{n}$ channels for HASH, and it significantly under-performed our proposed method. The two methods are proven to provide similar utility guarantees with similar complexities. In [15], only the first algorithm PEM1 is implemented, and no experimental comparison with SPM is conducted.

4.2.4 Concurrent work: PrivTrie.

We also found another concurrent work, PrivTrie [16], which considers a more realistic setting where the length of each user's value is not fixed. The main idea of PrivTrie is similar

to PEM and PEM1: For each location, it tests which prefix is more frequent, and then uses the frequent prefixes as the candidate to test the next location.

PrivTrie deals with ASCII string values and choose to extend by one character each time, this is equivalent to choosing $g = m/8$. This suggests that while researchers hit upon the idea of using PEM, how the parameter g impacts accuracy was not well understood, and researchers chose whatever g that appears natural.

The interesting idea (and the major structural difference from PEM) of PrivTrie is the dynamic group allocation. That it, less users will be allocated to estimate the very beginning prefixes, and more users can be assigned to evaluate the final estimation. Specifically, the users are partitioned into batches; each time, the aggregator estimates one prefix's frequency using one batch of users, until the aggregated count estimation exceeds a threshold $\frac{4n}{\epsilon\cdot\sqrt{\sum n_b}}$, where n_b is the number of users per batch, and $\sum n_b$ is the total number of users reporting this prefix. Then the prefix is considered frequent and pretended by one character. Each user can keep reporting on new prefixes so long as these prefixes are not result from extending those he already have reported. After identifying the frequent strings, a post-processing step modified from [17] is used to make the final estimation more accurate.

While the idea seems promising, PrivTrie is highly interactive. That is, the aggregator needs to explicitly tell each user which prefixes he needs to report. Note that PEM is non-interactive in a sense that each user can report his prefix once using OLH, and the aggregator can examine all prefixes of the same size.

Another shortcoming of PrivTrie is that it uses threshold testing. While the analysis is made easier than the top-k version, the empirical utility will be poor when the threshold is set to an inappropriate value. If the threshold is too low, there are too many prefix candidates, which will blow up the computational cost; if it is set too high, there will be very few, even no candidate identified. As can be seen in the evaluation section, the threshold value $\frac{4n}{\epsilon\cdot\sqrt{\sum n_b}}$ from paper [16] does not work in some of our experiment setting.

4.3 Choosing the Parameter g

An important parameter in PEM is g, the number of groups. As shown in Section 4.2.2, a smaller g value leads to higher computational cost. If we consider only computational efficiency, we would want to choose g to be $m - \lceil \log_2 k \rceil$. PEM1 does this. However, choosing a large g may not be good for accuracy. In this section, we answer the question how to choose g to obtain the best accuracy.

4.3.1 Impact of g

For a heavy hitter identification protocol to be accurate, we want two things:

- High recall: That is, truly frequent values will be identified as frequent.
- Low False Positive Rate (FPR): That is, few infrequent values will be identified as frequent.

The choice of g affects both recall and FPR in the following ways.

1. A larger g leads to lower recall because PEM filters candidate frequent prefixes g times. For a frequent value to be successfully identified, it needs to pass the filtering g times.
2. A larger g leads to lower recall and higher FPR because of its effect on sampling error. Since each group is a subset of the overall population, the distribution in a group may be different from the distribution of the population. With a larger g, the number of users in each group will be smaller, and the sampling error is larger.
3. A larger g leads to lower recall and higher FPR because of larger SD of the (normalized) estimation, which is linear in $\frac{1}{\sqrt{n/g}}$.
4. A larger g leads to lower FPR because in each iteration there are fewer candidates to test. Each candidate has some probability to get a very high estimated frequency. There are a fixed number of truly frequent candidates, the more candidates that are tested, the more false positives we will get.

Because g impacts accuracy in so many different ways, comparing overall impacts of different g values is quite challenging. We deal with the challenge by simplifying the comparison. We ignore the first and second effect, namely more rounds and larger sampling error, and focus on the third and fourth effect. That is, we compare the effect of smaller groups and fewer candidates. We will show that these two effects largely cancel each other out. Since the other factors all favor smaller g values, this shows that overall smaller g is preferred for accuracy. We will verify this using empirical evaluations in Section 4.3.4.

More specifically, we compare the accuracy of the last round only, assuming that the previous rounds have preserved all frequent prefixes and that the group for the last round has the same distribution as the overall population. That is, we perform a hypothetical comparison such as the following:

Example 3. *We want to comparing the effect of using $g = 4$ versus $g = 12$ when aiming to select the 2^8 frequent values with a 128-bit domain, with $n = 6 \times 10^6$ users. We want to know which of the following two hypothetical case leads to more accurate selections.*

- *Case $g = 4$: we have a group of $n/4 = 1.5m$ users, are given the set of 2^8 most frequent 98-bit prefixes, and are asked to select the 2^8 frequent values among the 2^{38} candidates.*
- *Case $g = 12$: we have a group of $n/12 = 0.5m$ users, and are given the set of 2^{118}-bit prefixes, and are asked to select the 2^8 frequent values among the 2^{18} candidates.*

In the above example, because the $g = 4$ case has a larger group size, this leads to a lower SD and more accurate estimates for the frequent values. However it has $2^{38} - 2^{18} \approx 2^{38}$ additional candidate values, which could lead to a lot more false positives. Our analysis focuses on understanding the FPR. We introduce the concept of sensitivity threshold.

4.3.2 The Sensitivity Threshold for LDP

The estimated frequency of a value v (computed by Equation (3.1) when OLH is used) is a linear function of the random variable I_v, which counts how many reports "support" v. Let n be the number of reports, then I_v is the summation of n Bernoulli random variables. Since n is very large in our setting (typically thousands or more), I_v can be approximated by

a Gaussian distribution. Now consider the estimated frequency of a value v, which is a linear function of I_v. We already know it is unbiased and its variance (e.g., Equation (3.12)); now we also know it follows a Gaussian distribution, and we can derive the mean and variance of the Gaussian distribution.

When one estimates the frequencies of all values in a large candidate set and then selects the k most frequent ones among them, one wants to avoid false positives that are caused by low-frequency values. Among a population of size n and domain of size d, where d is quite large, the vast majority of the values have a count of 0 or very close to 0. We thus care about when we make d independent samples from a zero-mean Gaussian distribution, how many sampled values will be above a certain threshold.

Definition 4.3.1 (Sensitivity Threshold)**.** *The threshold function $\Psi(\sigma, d, n, T)$ computes the smallest threshold θ such that from d independent sampling of a zero-mean Gaussian distribution with variance $\sigma^2 \cdot n$, the expected number of sampled values that are above $\theta \cdot n$ is $\leq T$.*

When using an FO with normalized standard deviation σ, if we want to take all values that have estimated frequency above θ, then we would expect to find around T false positives from the very low-frequency ones. Fixing the number of false positives one is willing to tolerate (the parameter T), one can reliably identify frequent values with frequency above $\theta = \Psi(\sigma, d, n, T)$. Specifically, assume the CDF that a Gaussian variable is sampled above $\theta \cdot n$ is

$$\Phi(\theta n|0, \sigma\sqrt{n}) = \int_{\theta n}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2 n}} \cdot e^{-\frac{x^2}{2\sigma^2 n}} dx$$

Ψ calculates a θ value such that the CDF equals T/d. Namely,

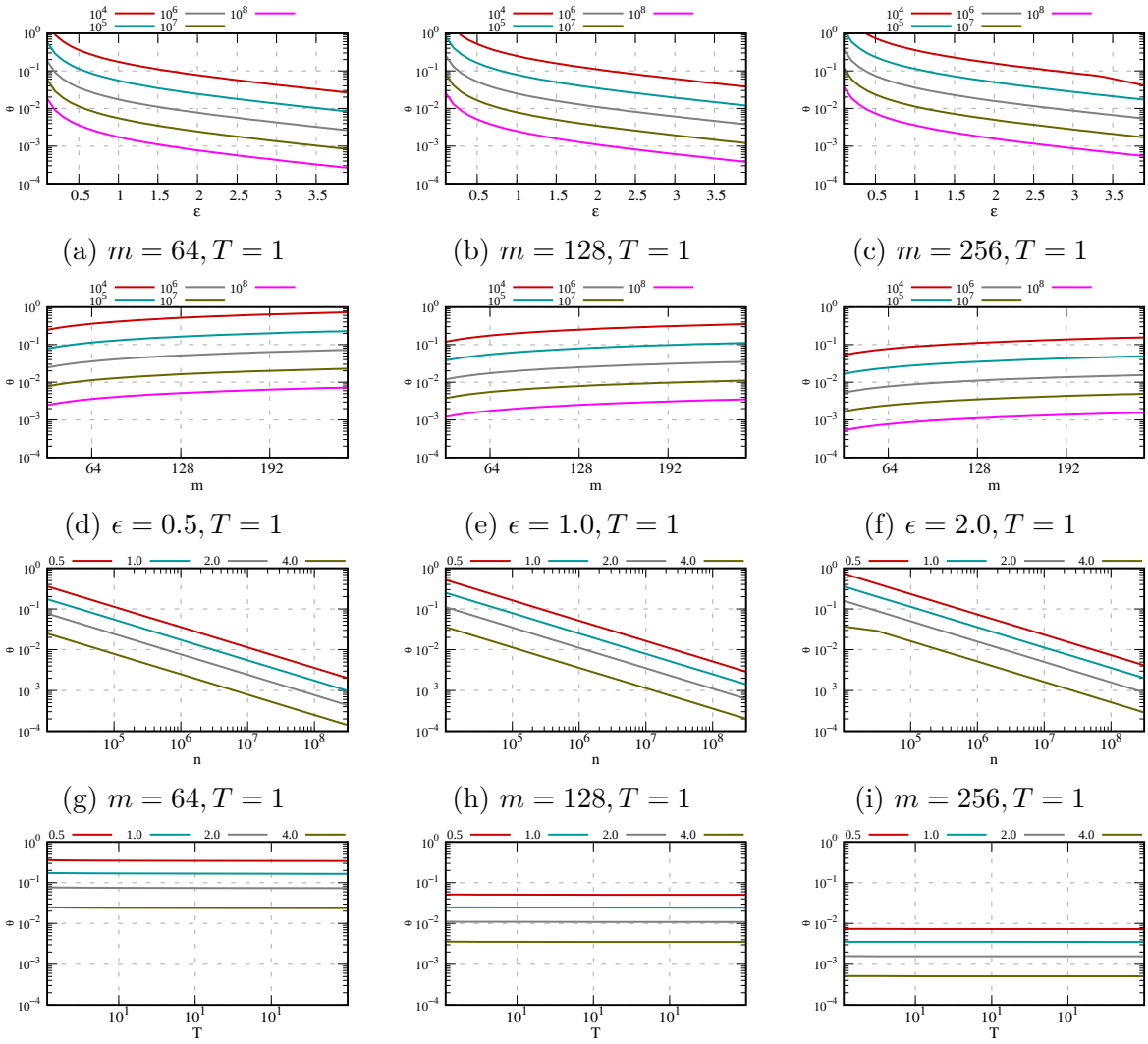
$$\Psi(\sigma, d, n, T) = \Phi^{-1}(T/d|0, \sigma\sqrt{n})$$

Two different g values lead to different n values and different d values; and we can thus compare their resulting θ values for the same t and σ. Assume OLH is used, $\sigma = \sqrt{\frac{4e^\epsilon}{(e^\epsilon-1)^2}}$ according to Equation (3.12). Figure 4.1 gives the value of $\Psi(\sigma, d, n, T)$ under different settings.

Table 4.1. The value of the threshold $\Psi(\sigma, 2^m, n, 1)$ for different ϵ, m, and n

ϵ, m, n \ g	1	2	3	4
$1, 32, 10^4$	0.119629	0.137374	0.144731	0.154686
$1, 32, 10^5$	0.037832	0.043285	0.045536	0.048738
$1, 32, 10^6$	0.011960	0.013674	0.014388	0.015396
$1, 32, 10^7$	0.003781	0.004322	0.004546	0.004866
$1, 48, 10^4$	0.149492	0.163341	0.168102	0.178922
$1, 48, 10^5$	0.047267	0.051595	0.052936	0.056182
$1, 48, 10^6$	0.014937	0.016314	0.016738	0.017751
$1, 48, 10^7$	0.004723	0.005158	0.005294	0.005610
$1, 64, 10^4$	0.174594	0.186712	0.192771	0.199696
$1, 64, 10^5$	0.055144	0.058866	0.060857	0.062761
$1, 64, 10^6$	0.017426	0.018599	0.019218	0.019845
$1, 64, 10^7$	0.005510	0.005881	0.006077	0.006271
$1, 128, 10^4$	0.250765	0.258555	0.261585	0.267211
$1, 128, 10^5$	0.079250	0.081804	0.082799	0.084400
$1, 128, 10^6$	0.025056	0.025844	0.026164	0.026683
$1, 128, 10^7$	0.007923	0.008173	0.008272	0.008437
$2, 32, 10^4$	0.053054	0.060933	0.064084	0.068286
$2, 32, 10^5$	0.016788	0.019178	0.020202	0.021646
$2, 32, 10^6$	0.005302	0.006063	0.006376	0.006825
$2, 32, 10^7$	0.001677	0.001916	0.002016	0.002158
$2, 48, 10^4$	0.066447	0.072487	0.074325	0.078790
$2, 48, 10^5$	0.020964	0.022907	0.023511	0.024903
$2, 48, 10^6$	0.006623	0.007235	0.007424	0.007865
$2, 48, 10^7$	0.002094	0.002287	0.002347	0.002488
$2, 64, 10^4$	0.077477	0.082466	0.085355	0.088244
$2, 64, 10^5$	0.024456	0.026111	0.026977	0.027844
$2, 64, 10^6$	0.007728	0.008248	0.008527	0.008800
$2, 64, 10^7$	0.002443	0.002608	0.002694	0.002781
$2, 128, 10^4$	0.111353	0.115030	0.116080	0.118706
$2, 128, 10^5$	0.035144	0.036247	0.036746	0.037508
$2, 128, 10^6$	0.011111	0.011463	0.011599	0.011835
$2, 128, 10^7$	0.003513	0.003624	0.003668	0.003741

Now we go back to the original problem of comparing the utility for different g values. Table 4.1 gives the calculated θ values for different settings. For $g > 1$, the aggregator estimates d values, where $d = k \cdot 2^{s_g - s_{g-1}} - k = k \cdot 2^{\lceil m - \lceil \log_2 k \rceil \rceil - \left\lceil (g-1) \cdot \frac{m - \lceil \log_2 k \rceil}{g} \right\rceil} - k$, using n/g users. In this table, each row corresponds to one setting, and each column corresponds to one g value. Note that within each row, the typical trend is that when g increases, θ increases.



(a) $m = 64, T = 1$ (b) $m = 128, T = 1$ (c) $m = 256, T = 1$

(d) $\epsilon = 0.5, T = 1$ (e) $\epsilon = 1.0, T = 1$ (f) $\epsilon = 2.0, T = 1$

(g) $m = 64, T = 1$ (h) $m = 128, T = 1$ (i) $m = 256, T = 1$

(j) $m = 64, n = 10^4$ (k) $m = 128, n = 10^6$ (l) $m = 256, n = 10^8$

Figure 4.1. Numerical results of Ψ.

4.3.3 Choice of g

Table 4.1 indicates that looking at the final test only, setting $g = 1$ (which equals OLH) makes identification of high-frequency items most reliable. In the context of heavy hitter identification, when taking the filtering effect of the previous rounds, the advantage of using smaller g values will be more pronounced.

Regarding the running time, on the other hand, a small g will make the running time increase greatly. In particular, should be as small as possible.

Table 4.2. The empirical utility (measured by the average number of identified heavy hitters) of PEM under different settings, assuming $m = 32$.

Dataset, ϵ \ g	1	2	3	4
URL, 1.0	6.4	4.7	4.1	4.0
URL, 2.0	14.3	12.8	11.4	8.5
AOL, 1.0	4.2	4.1	4.0	3.6
AOL, 2.0	8.5	7.8	6.9	7.2

Table 4.3. The empirical utility (measured by the average number of identified heavy hitters) of PEM under different settings, assuming $m = 32$. We assume that the candidates used in the last round are always the true heavy hitter prefixes.

Dataset, ϵ \ g	1	2	3	4
URL, 1.0	6.4	6.8	8.2	8.4
URL, 2.0	14.3	13.9	13.4	13.3
AOL, 1.0	4.2	4.6	5.0	5.2
AOL, 2.0	8.5	8.1	9.0	9.4

4.3.4 Verifying the Analytical Results Empirically

We compute the empirical results for the different g in Table 4.2 and Table 4.3. Under the same evaluation setup as in Section 4.4, the results are the average of 10 runs. Note that we only compute $m = 32$, because the computational time increases exponentially with m for $g = 1$. Specifically, in Table 4.2, we can see the performance is best when $g = 1$. In Table 4.3, we assume all the candidates for the last round are the ground truth. The setting for $g = 3$ is generally favorable, because the number of additional bits to be tested in the final round is smaller. Note that when $g = 1$, the result is slightly worse, because there are values with frequency close the that of the top-k heavy hitters, thus interfering the results (but this will not happen when m becomes larger). However, when $g > 1$, those values are to some extent eliminated. The standard deviations for both tables are around 1.

4.4 Evaluation

Now we discuss experiments that evaluate different protocols. Basically, we want to answer the following questions: First, how many heavy hitters can be effectively identified. Second, how much improvement is PEM over existing protocols. Finally, what are the effects of different design choices in PEM.

4.4.1 Evaluation Setup

Each experiment is run 10 times, and the average and standard deviation are reported. All algorithms are implemented in Python 2.7 and all the experiments are conducted on an Intel Core i7-4790 3.60GHz PC with 16GB memory.

Utility Metric

F1 and NCR scores are used to measure utility.

F-measure (F1). Define v_j as the j-th most frequent value. Denote ground truth for top k values as $C_T = \{v_1, v_2, \ldots, v_k\}$. The k values identified by the protocol is C_g. $C_T \cap C_g$ is the set of real top-k values that are identified by the protocol, and $C_T \cup C_g$ is the union of the two sets. We use the widely used F-measure, which is the harmonic mean of precision and recall, i.e.,

$$\text{F1} = \frac{2}{1/P + 1/R} = \frac{2PR}{P+R}$$
$$\text{where } P = \frac{|C_T \cap C_g|}{|C_g|}, R = \frac{|C_T \cap C_g|}{|C_T|}$$

We note that when $|C_T| = |C_g|$, the precision P equals the recall R, and the F-measure equals the precision, as well as 1 minus the false negative rate.

Normalized Cumulative Rank (NCR). The F-measure uses only the unordered set C_T as the ground truth. As a result, missing the value with the highest frequency is penalized the same as missing any others. To address this limitation, we assign a quality function $q(\cdot)$ to each value, and use the Normalized Cumulative Gain (NCG) metric:

$$\text{NCG} = \frac{\sum_{v \in C_g} q(v)}{\sum_{v \in C_T} q(v)}$$

We instantiate the quality function using v's rank as follows: the highest ranked value has a score of k (i.e., $q(v_1) = k$), the next one has score $k-1$, and so on; the k-th value has a score of 1, and all other values have scores of 0. To normalize this into a value between 0 and 1, we divide the sum of scores by the maximum possible score, i.e., $\frac{k(k+1)}{2}$. This gives rise to what we call the Normalized Cumulative Rank (NCR); this metric uses the true rank information of the top-k values.

Both F-measure and NCR are in the range $[0.0, 1.0]$, where higher values indicate better accuracy. We present results using these metrics and observe that the correlation among them is quite stable.

Dataset

The following three datasets are used. We assume the Zipf's distribution when optimizing PEM. Note that in the real world, auxiliary information (heavy hitter dictionary) may exist to help improve the result. For example, the system BLENDER [18] is proposed to work under the assumption that a certain amount of users will participate in a centralized DP protocol to find out the dictionary of heavy hitters. However, our focus is on the case where there is no additional dictionary or the heavy hitters are changing frequently so that existing dictionaries are not reliable to provide up-to-date information.

Frequent URL. In Rappor [14], the authors synthesized one million urls from a confidential distribution of only 100 websites. The urls are fixed to be 20 bytes (160 bits) long (padding or truncating if needed). We do not have the same dataset but collect a similar dataset from Quantcast [19]. The dataset contains domain name and monthly visited people of the 80 thousand most frequently visited websites. We limit urls to 20 bytes and limit the analysis to a 5-minute period (we downloaded the data for one month, which contains 10 billion clicks, and down-sample it for 5-minute, i.e., divide the count for each number by

$30 \cdot 24 \cdot 12$), resulting a dataset containing 1.2 million data points, and 27 thousand unique urls.

Query Trends. The AOL dataset contains user queries on AOL website during the first three months in 2006. Similar to the settings of [18], we assume each user reports one query (wlog., the first query). The queries are limited to be 6 bytes long. This results a dataset of around 0.5 million queries including 0.12 million unique ones.

Synthetic Dataset. We generate a synthetic dataset of $n = 1000000$ data points following the exponential distribution (also known as geometric distribution). The values (heavy hitters) are randomly distributed. Each value is represented by $m = 64$ bits. The exponential scale is 0.05, which is close to the experimental setting in [2].

Competitors

We initialize PEM so that the overall number of queries to the frequency oracle is bounded to 2^{20}. In practice, PEM runs fast. We also consider the following algorithms: PEM, PEM1, HASH, and SPM.

PEM1, SPM and HASH were designed to find heavy hitters based on threshold, but PEM works for top k heavy hitters. For a fair comparison, we make some changes described below. Note that PEM can also be changed to work for threshold. The corresponding results are shown in Section 4.4.2.

1. Replace Threshold Test. Existing methods require internal test and filtering based on a threshold. That is, there is a final testing phase on all the identified values. Only those tested above a threshold will be returned. We replace this constraint by releasing the top k values for a fair comparison.

Specifically, in PEM1, at each iteration, prefixes with frequency lower than a fixed threshold is discarded. We keep k values at each iteration. In Rappor, each segment or segment-pair will be identified if its frequency is estimated above a threshold. We relax this by limiting exactly k patterns in each segment. This ensures to identify at least k heavy hitters. For the location-pairs, we keep adding segment-pairs until more than k candidates are identified.

2. Same Frequency Oracle. Existing methods use non-optimal LDP primitives, but they can be changed. Specifically, SPM use RAP [2] as the internal LDP primitive, and HASH use BLH, and PEM1 uses counter median sketch based primitive. We replace them with OLH.

3. Reduce Number of Groups. For the url dataset, SPM specifies one segment length to be two bytes. But for other domain length, there is no clear specification as to how long each segment should be. Guided by the analysis of Section 4.3, we make the segment length as long as possible, conditioned on that they do not take too much more computational time than PEM.

HASH uses $n^{1.5}$ channel, which is infeasible in many scenarios. We observe that collision of other non-frequent values does not effect much, and propose to use $k^{1.5}$ channels. This is similar to the second protocol described in [15].

4.4.2 Detailed Results

Effect of ϵ

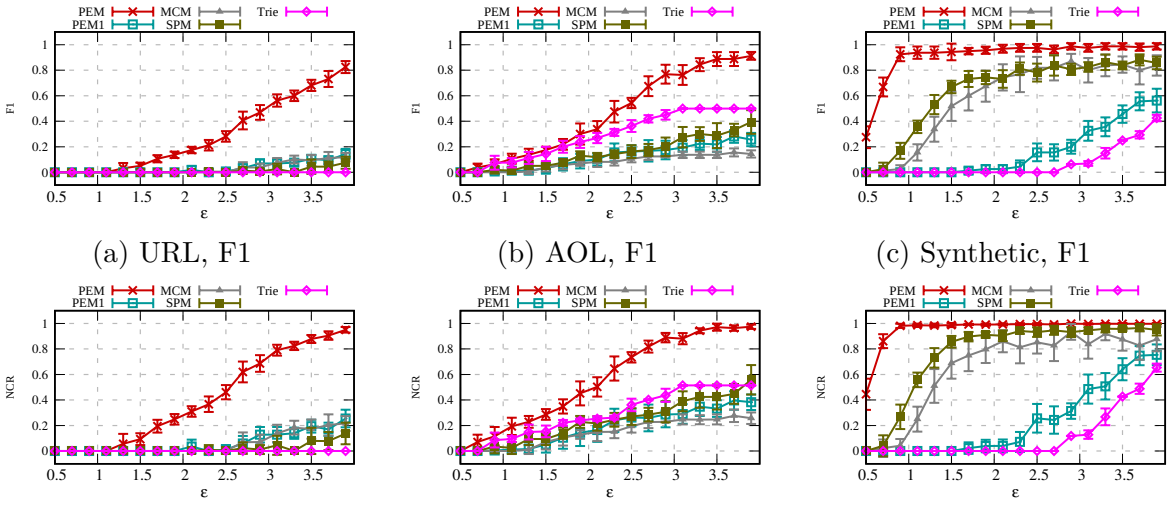
We show F1 and NCR results of different methods varying ϵ in Figure 4.2. It is clear that PEM performs best among all three protocols. When ϵ increases, the number of heavy hitters that can be identified will increase. The improvement is more significant when ϵ is larger.

When $\epsilon = 4$, PEM achieves F1 $= 0.9$, meaning that more than ten frequent URLs can be identified; on the other hand, other methods can only identify two.

Effect of k

Figure 4.3 gives F1 and NCR results of different methods varying k. Similarly, we can see that PEM outperforms its competitors. Note the correlation between F1 and NCR is close: a protocol with better F1 score will also have a better NCR score. Thus, from now on, we ignore the NCR scores.

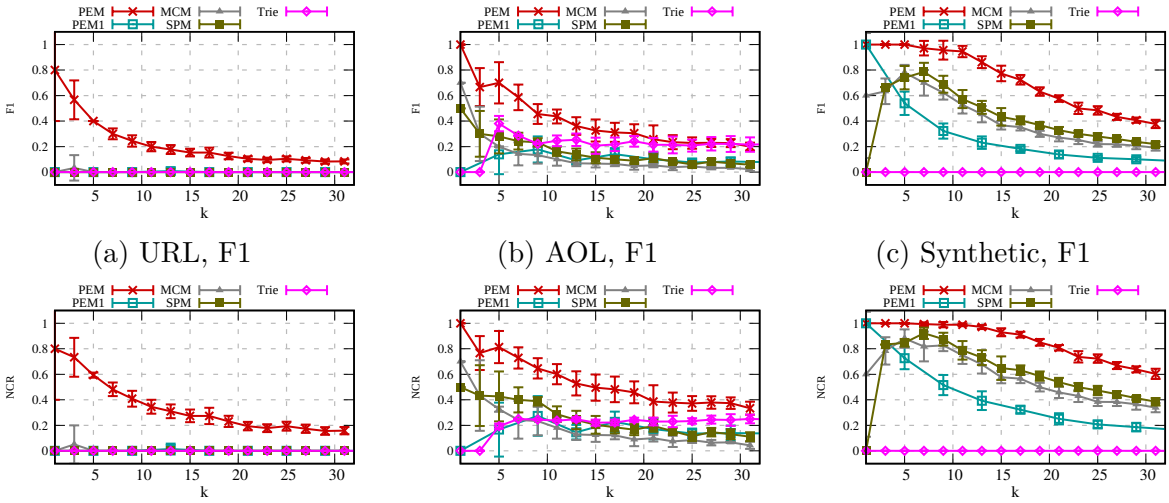
For most of the cases, utility scores decrease with k, since the less frequent values are harder to identify. In the synthetic dataset, PEM achieves almost full utility for up to $k = 30$.



(a) URL, F1 (b) AOL, F1 (c) Synthetic, F1

(d) URL, NCR (e) AOL, NCR (f) Synthetic, NCR

Figure 4.2. Evaluation of the datasets, vary ϵ while fixing $k = 16$.

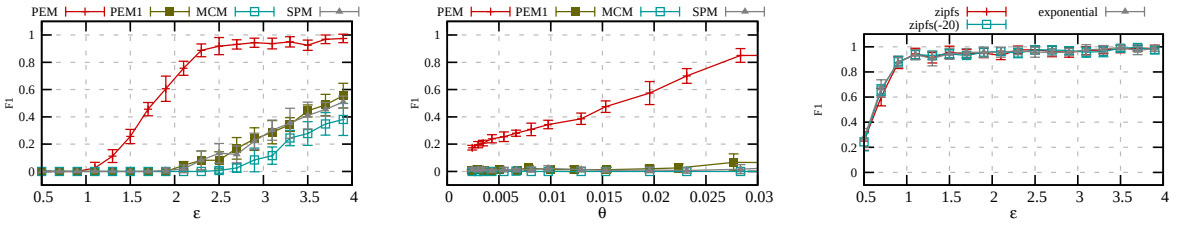


(a) URL, F1 (b) AOL, F1 (c) Synthetic, F1

(d) URL, NCR (e) AOL, NCR (f) Synthetic, NCR

Figure 4.3. Evaluation of the datasets, varying k while fixing $\epsilon = 2$.

On the other hand, in some cases, as k increases, the absolute number of heavy hitters that can be identified stops increasing. This is because the task becomes hard so that even with more guesses, it is still hard to find the heavy hitters.



(a) F1, vary ϵ fixing $\theta = 0.023$ (b) F1, vary θ fixing $\epsilon = 2$ (c) F1, vary θ fixing $\epsilon = 4$

Figure 4.4. Evaluation of the synthetic datasets, vary one of ϵ and θ while fixing the other. $m = 64, n = 1000000$.

Evaluation of Threshold Version

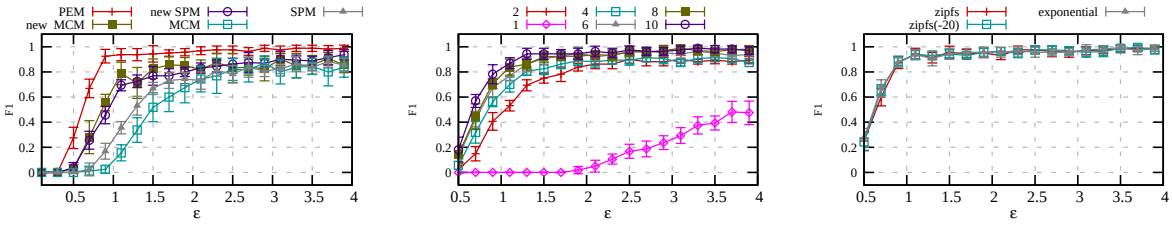
In this section, we modify PEM in order to identify heavy hitters with frequencies above a threshold θ. Note that each threshold value θ can be translated into a corresponding k value. The lower the θ, the bigger the k is.

Similar to the previous section, we also show results varying ϵ and θ. For brevity, we only show F1 for the Exponential dataset in Figure 4.4. The results are similar in other datasets.

As can be seen from Figure 4.4a, when we fix $\theta = 0.023$ (0.023 is around frequency of the 16-the most frequent value in the dataset), PEM performs better. This advantage is most profound when $\epsilon = 2$, where PEM achieves performs much better than existing methods. The effects of fixing ϵ and varying θ are also demonstrated in Figure 4.4b and 4.4c.

Effect of Partitioning Users

We further improve HASH and SPM algorithms so that instead of split privacy budget, we allocate 10% of users for the final testing. The result shown in Figure 4.5a demonstrates the advantage of partitioning users. Especially, when $\epsilon = 1.2$, the original HASH method achieves F1 less than 0.2, while the new version achieves nearly 0.8. For brevity, we only show F1 score on Exponential dataset, but the trend is similar in other settings. Note PEM1 already partition the users.



(a) Effect of partitioning users.

(b) Effect of Δs.

(c) Effect of distribution assumption.

Figure 4.5. Evaluation of the synthetic datasets, vary ϵ. $m = 64, n = 1000000$. F1 is plotted.

Effect of g

In Figure 4.5b, we demonstrate the effect of group g. We set g so that the difference between any two consecutive prefixes s_i and s_{i-1}, defined as Δs, is 1 (PEM1), $2, 4, 6, 8, 10$ and plot the results. It is clear that when Δs increases, the overall utility is better. When $\epsilon = 0.9$, we see the F1 score is 0.4 when $\Delta s = 2$, and 0.8 when $\Delta s = 10$. Note that there should be a limit on how large Δs can be, that is, Δs is limited by the number of queries the aggregator can make.

Comparison of Estimation Accuracy

Having demonstrated that PEM achieves better utility (no matter F1 or NCR scores), we compare the estimation accuracy. We use the *Mean Squared Error* (Var) as the metric, that is,

$$\text{Var} = \frac{1}{|C_T \cap C_g|} \sum_{v \in C_T \cap C_g} \left(n_v - \hat{f}(v)\right)^2,$$

where n_v is the true count of v and $\hat{f}(v)$ is its estimation by the protocol. As the quantity converges to the theoretical variance, we also use Var to denote it Note that we only account heavy hitters that are successfully identified by the protocol, i.e., $v \in C_T \cap C_g$.

Figure 4.6 shows comparison of estimation variance for different methods. Observe that the HASH method has smaller variance than Rappor, because the final testing step of HASH

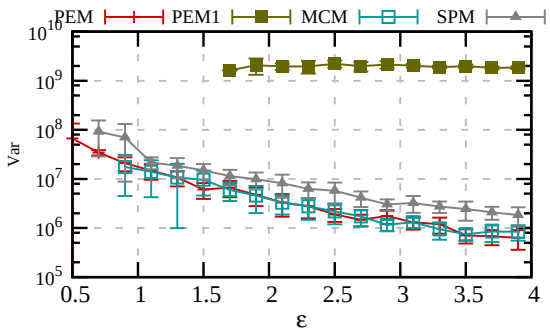


Figure 4.6. Evaluation of the synthetic datasets, vary ϵ. $m = 64, n = 1000000$.

uses half of the ϵ, while that of Rappor uses one third. As a comparison, PEM uses only the last group, which is one sixth of users, and achieves similar estimation accuracy. PEM1 uses smaller amount of users, thus produces higher variance.

Effect of Distribution Assumption

In the experiment, to mimic the blindness of the distribution, we use a Zipf's distribution to optimize PEM. Note that in practice, it is hard to know the real distribution of the dataset. The task of getting an accurate distribution is therefore left to the practitioners. Here, we argue that except in extreme cases, the influence of a poor assumption to the final result is not much. As we can see from Figure 4.5c, under different assumptions, the results are very similar.

Comparison with PrivTrie

In [16], the authors compare PEM, setting $\Delta s = 5$ (Δs is the difference between any two consecutive prefixes s_i and s_{i-1}). We believe they did not conduct a fair comparison. So we redo the experimental comparison.

In the experiment, we assume that all the values are binary and have the same length m, and the size of the alphabet in PrivTrie is 2^8. The batch size is $\max(n/1000, 800/\epsilon^2)$. For a fair comparison, we do not include the final consistent step proposed in [16], because this

is orthogonal to the structure of the method, and all methods can potentially benefit from this additional step.

The results are shown in Figure 4.2 and 4.3. As can be seen, the utility of PrivTrie is always worse than PEM. Specifically, for the URL dataset, even when $\epsilon = 4$, PrivTrie still identifies no meaningful heavy hitters. As a contrast, PEM achieves F1 > 0.8 when $\epsilon = 4$. The results are as expected from the analysis in Section 4.2.4. Note that PrivTrie is highly interactive in that each user in PrivTrie will be contacted multiple times; and the communication cost is high. In contrast, existing methods are all non-interactive.

5. FREQUENT ITEMSET MINING

(A version of this chapter has been previously published in IEEE SP 2018 [20].)

In [21], Qin et al. considered the problem of locally differentially private frequent itemset mining. In this setting, there is a fixed set I of items. Each user j's value $\mathbf{v}^j$ is a set (we also call it a transaction), and $\mathbf{v}^j \subseteq I$. For any item $x \in I$, its frequency is defined as the number of transactions that include x, i.e., $f_x \coloneqq |\{\mathbf{v}^j \mid x \in \mathbf{v}^j\}|$. Similarly, the frequency of any itemset $\mathbf{x} \subseteq I$ is defined as the number of transactions that include $\mathbf{x}$ as a subset, i.e., $f_{\mathbf{x}} \coloneqq |\{\mathbf{v}^j \mid \mathbf{x} \subseteq \mathbf{v}^j\}|$. The goal in this setting is to find items and, more generally, itemsets that are frequent in the population. An item (itemset) is a top-k frequent item (itemset) if its frequency is among the k highest for all items (itemsets).

We want to provide the strong notion of local differential privacy (LDP), namely, for any pair of possible itemsets $\mathbf{v}$ and $\mathbf{v}'$, their output distributions are similar (with respect to ϵ).

This problem is quite challenging even when one just tries to find frequent items. Encoding each transaction as a single value in the domain $D = \mathcal{P}(I)$ (i.e., D is the power set of I), and using existing FO protocols does not work. While we described methods for large domains in last chapter, such techniques still does not scale to the case where the binary encoding of the input domain has more than a few hundred bits. We want to be able deal with hundreds or thousands of items. An FO protocol can identify only values that are very frequent in the population, because the scale of the added noises is linear to square root of the population size [13]. It is quite possible that each particular transaction appears relative infrequently, even though some items and itemsets appear very frequently. When no value in $\mathcal{P}(I)$ is frequent enough to be identified, using a direct encoding an aggregator can obtain only noises.

For example, assume we have five transactions $\{a, c, e\}, \{b, d, e\}, \{a, b, e\}, \{a, d, e\}, \{a, f\}$. While no transaction appears more than once, items a and e each appears 4 times, and the itemset $\{a, e\}$ appears 3 times. Thus the three most frequent itemsets are $\{a\}, \{e\}, \{a, e\}$.

5.1 Existing Work

5.1.1 LDPMiner

To the best of our knowledge, LDPMiner [21] is the only protocol for dealing with set values in the LDP setting. While finding frequent itemsets is a natural goal, LDPMiner finds only frequent items (i.e., singleton itemsets) and leaves the frequent itemset mining as an open problem. LDPMiner has two phases.

Phase 1: Candidate Set Identification. The goal of Phase 1 is to identify a candidate set for frequent items. The protocol requires as input a parameter L, which is the 90th percentile of transaction lengths . That is, about 90% of all transactions have length no more than L. When L is not known, it needs to be estimated. In [21], it is assumed that L is available.

In Phase 1, each user whose transaction $\mathbf{v}$ has less than L items first pads it with dummy items so that the transaction has size L. Then, the user selects at uniform random one item v from the padded transaction (which could result in a dummy item), and uses FO to report it with privacy budget $\epsilon/2$. That is, each user sends to the aggregator $\mathcal{A}_{\mathsf{FO}(\epsilon/2)}(v)$, where $\mathcal{A}_{\cdot}(\cdot)$ denotes the encode-perturbation algorithm of any LDP scheme described in Chapter 3. Note that the FO can perturb the original value into any value including the dummy item.

The aggregator then computes, for each item $x \in I$, its estimated frequency as

$$\hat{f}_{\mathsf{FO}(\epsilon/2)}(x) \cdot L$$

where $\hat{f}_{\cdot}(\cdot)$ denotes any FO's aggregation function described in Chapter 3. The intuition behind the above estimation is that in each transaction of length L, each item x will be selected and reported with probability $\frac{1}{L}$. Hence one needs to multiply the frequency oracle's estimation by a factor of L. Since 90% of transactions will have length exactly L after padding, this estimation is reasonably accurate. From the estimates, the aggregator identifies

S, the set of $2k$ items that have the highest estimated frequencies, and sends S to the users. Size of S is set to be twice that of the goal so that few candidates are missed in this step.

Phase 2: Frequency Estimation. On receiving S, each user intersects it with $\mathbf{v}$, which results in a transaction of length no more than $|S| = 2k$. She then pads her transaction $\mathbf{v} \cap S$ to be of size $2k$, selects at uniform random one item v from the padded transaction, and sends $\mathcal{A}_{\mathsf{FO}(\epsilon/2)}(v)$ to the aggregator. Since each user sends two things, each in a way that satisfies $(\epsilon/2)$-LDP, by sequential composition, the protocol satisfies ϵ-LDP.

The aggregator estimates frequency for each item $x \in S$:

$$\hat{f}_{\mathsf{FO}(\epsilon/2)}(x) \cdot 2k$$

Since the size of all user's transactions have size $2k$ after padding, the estimated frequencies are unbiased.

5.2 Padding-and-Sampling-based Frequency Oracles

The LDPMiner protocol deals with the challenge of set-valued inputs by using padding and sampling before applying an FO protocol to report. We call such protocols *Padding-and-Sampling-based Frequency Oracle (*PSFO*)* protocols. They use a padding-and-sampling function, defined as follows.

Definition 5.2.1 (PS)**.** *The padding and sampling function* PS *is specified by a positive integer ℓ and takes a set $\boldsymbol{v} \subseteq I$ as input. It assumes the existence of ℓ dummy items $\perp_1, \perp_2, \ldots, \perp_\ell \notin I$. $\mathsf{PS}_\ell(\boldsymbol{v})$ does the following: If $|\boldsymbol{v}| < \ell$, it adds $\ell - |\boldsymbol{v}|$ different random dummy elements to $\boldsymbol{v}$. It then selects an element v at uniform random and outputs that element.*

A PSFO protocol then uses an FO protocol to transmit the element v. Note that the domain of the FO becomes $I \cup \{\perp_1, \perp_2, \ldots, \perp_\ell\}$. To estimate the frequency of an item x, one obtains the frequency estimation of x from the FO protocol, and then multiplies it by ℓ. More formally,

Definition 5.2.2 (PSFO)**.** *A padding-and-sample-based frequency oracle (*PSFO*) protocol is specified by three parameters: a positive integer ℓ, a frequency oracle* FO*, and the privacy budget ϵ. It is composed of a pair of algorithms: $\langle \mathcal{A}, \hat{f} \rangle$, defined as follows.*

$$\mathsf{PSFO}(\ell, \mathsf{FO}, \epsilon) \coloneqq \langle \mathcal{A}_{\mathsf{FO}(\epsilon)}(\mathsf{PS}_\ell(\cdot)), \quad \hat{f}_{\mathsf{FO}(\epsilon)}(\cdot) \cdot \ell \rangle$$

Note that if one does not do the padding step, it is equivalent to setting $\ell = 1$. Doing so significantly under-estimates the true counts. With padding to length ℓ and then sampling, one can multiply the estimated counts by ℓ to correct the under estimation. However, items that appear in transactions longer than ℓ can still be underestimated. At the same time, multiplying the estimation by ℓ will enlarge any error due to noise by a factor of ℓ.

Using this notation, the two phases of LDPMiner can be cast as using $\mathsf{PSFO}(L, \mathsf{FO}, \epsilon/2)$ in Phase 1 and $\mathsf{PSFO}(2k, \mathsf{FO}, \epsilon/2)$ in Phase 2.

5.2.1 Privacy Amplification of GRR

LDPMiner uses the FO protocol in a black-box fashion. That is, in order to satisfy ϵ-LDP, it invokes the FO protocol with the same privacy parameter ϵ. We observe that, since the sampling step randomly selects an item, it has an amplification effect for privacy. This effect has been observed and studied in the standard DP setting [22]: If one applies an algorithm to a dataset randomly sampled from the input with a sampling rate of $\beta < 1$, to satisfy ϵ-DP, the algorithm can use a privacy budget of ϵ' such that $\frac{e^{\epsilon'}-1}{e^{\epsilon}-1} = \frac{1}{\beta}$.

We observe that the same privacy amplification effect exists when using the Generalized Random Response (GRR) in PSFO.

Theorem 5.2.1 (PSFO-GRR: Privacy Amplification)**.** $\mathcal{A}_{\mathsf{GRR}(\epsilon')}(\mathsf{PS}_\ell(\cdot))$ *satisfies ϵ-LDP, such that $\epsilon' = \ln(\ell \cdot (e^{\epsilon} - 1) + 1)$.*

Proof: Let $d' = |I| + \ell$ be the size of the new domain ($I' = I \cup \{\perp_1, \ldots, \perp_\ell\}$), ϵ' as the privacy budget used in GRR. As defined in Equation (3.4), we have $p' = \frac{e^{\epsilon'}}{e^{\epsilon'}+d'-1}$ and $q' = \frac{1}{e^{\epsilon'}+d'-1}$ as the perturbation probabilities.

It suffices to prove that for any $\epsilon \geq 0$, any $\mathbf{v}_1, \mathbf{v}_2 \subseteq I$, and any possible output $t \in I'$, $\frac{p_1}{p_2} \leq e^{\epsilon}$, where

$$p_1 = \mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon')}(\mathsf{PS}_\ell(\mathbf{v}_1)) = t\right], \text{ and}$$
$$p_2 = \mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon')}(\mathsf{PS}_\ell(\mathbf{v}_2)) = t\right].$$

We first examine p_1. When $t \in \mathbf{v}_1$,

$$\begin{aligned}
p_1 =& \mathsf{Pr}\left[t \text{ is sampled}\right] \cdot p' + \mathsf{Pr}\left[t \text{ is not sampled}\right] \cdot q' \\
=& \frac{1}{\max\{|\mathbf{v}_1|, \ell\}} \cdot p' + \frac{\max\{|\mathbf{v}_1|, \ell\} - 1}{\max\{|\mathbf{v}_1|, \ell\}} \cdot q' \\
=& q' + \frac{1}{\max\{|\mathbf{v}_1|, \ell\}} \cdot (p' - q') \\
\leq& q' + \frac{1}{\ell} \cdot (p' - q') \\
=& \frac{1}{\ell} p' + \frac{\ell - 1}{\ell} q'
\end{aligned}$$

When $t \notin \mathbf{v}_1$, $p_1 = q'$. Similarly, for p_2, when $t \in \mathbf{v}_2$,

$$\begin{aligned}
p_2 =& \mathsf{Pr}\left[t \text{ is sampled}\right] \cdot p' + \mathsf{Pr}\left[t \text{ is not sampled}\right] \cdot q' \\
=& \mathsf{Pr}\left[t \text{ is sampled}\right] \cdot (q' + p' - q') \\
&+ \mathsf{Pr}\left[t \text{ is not sampled}\right] \cdot q' \\
=& q' + \mathsf{Pr}\left[t \text{ is sampled}\right] \cdot (p' - q') \geq q'
\end{aligned}$$

When $t \notin \mathbf{v}_2$, $p_2 = q'$. Thus $\frac{p_1}{p_2}$ is maximized when $p_1 = \frac{1}{\ell} p' + \frac{\ell - 1}{\ell} q'$ and $p_2 = q'$. That is,

$$\begin{aligned}
\frac{p_1}{p_2} &\leq \frac{p'/\ell + q'(\ell - 1)/\ell}{q'} \\
&\leq e^{\epsilon'} \frac{1}{\ell} + \frac{\ell - 1}{\ell} \\
&= \frac{(\ell \cdot (e^{\epsilon} - 1) + 1)}{\ell} + \frac{\ell - 1}{\ell} = e^{\epsilon}
\end{aligned}$$

Table 5.1. Numerical value of ϵ' under different ϵ and ℓ.

ϵ \ ℓ	2	5	10	20	50	100
0.1	0.19	0.42	0.72	1.13	1.83	2.44
0.5	0.83	1.45	2.01	2.64	3.51	4.19
1.0	1.49	2.26	2.90	3.57	4.46	5.15
2.0	2.62	3.49	4.17	4.86	5.77	6.46
4.0	4.68	5.59	6.29	6.98	7.89	8.59

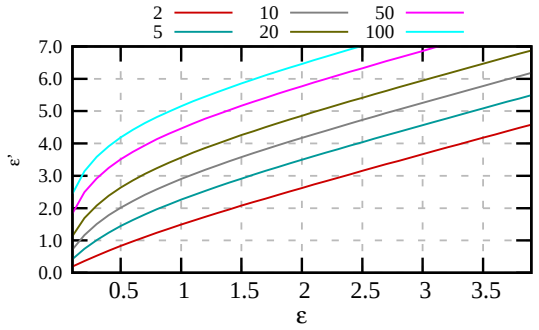


Figure 5.1. Privacy amplification effect for different ℓ.

□

Approximately, the privacy budget will be amplified by a factor of $\ln \ell$ (will be the same if $\ell = 1$). Table 5.1 and Figure 5.1 give the corresponding ϵ' value for ϵ under different ℓ.

5.2.2 No Privacy Amplification of other FO

Interestingly, we found that this privacy amplification effect does not exist for OLH. The reason is that, in GRR, the output domain of the perturbation is the same as the input domain; thus each reported value y can "support" a single input element $x = y$ in I. In OLH, however, the reported value takes the form of $\langle H, j \rangle$ and can support any element x in I such that $H(x) = j$. In case the chosen hash function H hashes all the user's items into the same value, no matter how we sample, the hashed result after sampling will always be the same value. Therefore, there is no privacy amplification in the sampling.

Theorem 5.2.2 (PSFO-OLH: No Privacy Amplification)**.** $\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_\ell(\cdot))$ *does not satisfy* ϵ*-LDP for any* $\epsilon < \epsilon'$ *when the input domain* I *is sufficiently large.*

Proof: Let g be the output size of hash functions. Consider an input domain I such that $|I| \geq g\ell + 1$. Let H be the chosen hash function. By the pigeon hole principle, there exists a value y such that H hashes at least ℓ items into y. Let $\mathbf{v}_1$ consists of ℓ items that are hashed to y, and $\mathbf{v}_2$ consists of items that are not hashed to y. Then

$$\begin{aligned}
&\frac{\mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon)}(\mathsf{PS}_\ell(\mathbf{v}_1)) = \langle H, y\rangle\right]}{\mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon)}(\mathsf{PS}_\ell(\mathbf{v}_2)) = \langle H, y\rangle\right]} \\
=&\frac{\mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon)}(H(\mathsf{PS}_\ell(\mathbf{v}_1))) = y | H\right]}{\mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon)}(H(\mathsf{PS}_\ell(\mathbf{v}_2))) = y | H\right]} \\
=&\frac{\mathsf{Pr}\left[H \text{ is picked}\right] \cdot p'}{\mathsf{Pr}\left[H \text{ is picked}\right] \cdot q'} = \frac{p'}{q'} = e^{\epsilon'}
\end{aligned}$$

Therefore, $\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_\ell(\cdot))$ is not ϵ-LDP for any $\epsilon < \epsilon'$. □

In Chapter 3, another FO protocol, Optimal Unary Encoding (OUE), was proposed. It has similar accuracy as OLH. In OUE, the reported value is a binary vector, each bit representing one possible input value. One reported value can have multiple bits being 1, supporting multiple input values. Similar to OLH, in case the reported vector supports all the user's items, there is no privacy amplification.

Note that from each user's point of view, the hash function H is randomly chosen. Thus only when the user happens to choose a hash function H that hashes all the users' items into the same hash value, would there be no privacy amplification benefit at all. However, this can happen with only small probability. This observation suggests that (ϵ, δ)-LDP can be applied to obtain some amplification effect.

5.2.3 Utility of PSFO

We now analyze the accuracy of PSFO. We first show that PSFO is unbiased when each user's itemset size is no more than ℓ.

Theorem 5.2.3 (PSFO Expectation)**.** $\mathsf{PSFO}(\ell, \mathsf{FO}, \epsilon)$ *is unbiased when* $\ell \geq \max_{j\in[n]} |\boldsymbol{v}^j|$. *That is,*

$$\mathrm{E}\left[\hat{f}_{\mathsf{PSFO}(\ell,\mathsf{FO},\epsilon)}(x)\right] = n_x,$$

where n_x *is the number of users who have item* x.

Proof: We prove for GRR, using the general aggregate function given in Equation (3.1). The proof for OLH can also be derived similarly.

$$\begin{aligned}
&\mathrm{E}\left[\hat{f}_{\mathsf{PSFO}(\ell,\mathsf{GRR},\epsilon)}(x)\right] = \mathrm{E}\left[\hat{f}_{\mathsf{GRR}(\epsilon')}(x)\cdot\ell\right] = \mathrm{E}\left[\frac{C(x)-nq'}{p'-q'}\cdot\ell\right] \\
=&\ell\cdot\frac{n_x\frac{1}{\ell}p' + n_x\frac{\ell-1}{\ell}q' + (n-n_x)q' - nq'}{p'-q'} = \ell\cdot\frac{n_x\frac{1}{\ell}(p'-q') + n_xq' + (n-n_x)q' - nq'}{p'-q'} = n_x
\end{aligned}$$

□

The estimation is inherently noisy. We now calculate the variance of the estimation.

Theorem 5.2.4 (PSFO Variance)**.** $\mathsf{PSFO}(\ell, \mathsf{FO}, \epsilon)$ *has variance* ℓ^2 *times that of* FO *when* $\ell \geq \max_{j\in[n]} |\boldsymbol{v}^j|$. *That is,*

$$\mathrm{Var}[\hat{f}_{\mathsf{PSFO}(\ell,\mathsf{FO},\epsilon)}(x)] = \ell^2\cdot\mathrm{Var}[\hat{f}_{\mathsf{FO}(\epsilon')}(x)],$$

where $\epsilon' = \ln(\ell\cdot(e^\epsilon-1)+1)$ *if* FO *is* GRR.

Proof: We prove for GRR, and the proof for OLH can be easily derived.

$$\begin{aligned}
&\mathrm{Var}[\hat{f}_{\mathsf{PSFO}(\ell,\mathsf{GRR},\epsilon)}(x)] = \mathrm{Var}[\hat{f}_{\mathsf{GRR}(\epsilon')}(x)\cdot\ell] \\
=&\mathrm{Var}\left[\frac{C(x)-nq'}{p'-q'}\cdot\ell\right] = \frac{\ell^2}{(p'-q')^2}\cdot\sum_j\mathrm{Var}[C(x)] \\
=&\frac{\ell^2}{(p'-q')^2}\cdot\left[n_x\left(\frac{1}{\ell}p'+\frac{\ell-1}{\ell}q'\right)\left(1-\left(\frac{1}{\ell}p'+\frac{\ell-1}{\ell}q'\right)\right)\right. \\
&\left.+(n-n_x)q'(1-q')\right] \\
\simeq&\frac{\ell^2}{(p'-q')^2}\cdot[nq'(1-q')] = \ell^2\cdot\mathrm{Var}[\hat{f}_{\mathsf{GRR}(\epsilon')}(x)]
\end{aligned}$$

□

5.2.4 Adaptive FO

PSFO needs to use an FO protocol. In Chapter 3, it was shown that one should choose GRR when $d < 3e^\epsilon + 2$ (where $d = |D|$ is the size of the domain under consideration), and OLH otherwise. With sampling, GRR can benefit from privacy amplification, but OLH benefit less. As a result, the criterion for choosing between GRR and OLH changes. For GRR, when ϵ is used in PSFO, the effective privacy budget GRR can use becomes $\ln(\ell(e^\epsilon - 1) + 1)$. We use (3.5) (with domain size $|I'| = d + \ell$) and get:

$$\begin{aligned}
&\mathrm{Var}[\hat{f}_{\mathsf{GRR}(\ln(\ell(\mathsf{e}^\epsilon-1)+1)}(x) \cdot \ell] \\
=&n \cdot \ell^2 \cdot \frac{d + l - 2 + \ell \cdot (e^\epsilon - 1) + 1}{(\ell \cdot (e^\epsilon - 1) + 1 - 1)^2} \\
=&n \cdot \frac{d + l - 1 + \ell \cdot (e^\epsilon - 1)}{(e^\epsilon - 1)^2} \\
=&n \cdot \frac{e^\epsilon \cdot \ell + d - 1}{(e^\epsilon - 1)^2}
\end{aligned} \tag{5.1}$$

For OLH, by Equation (3.12) we have variance independent on d:

$$\mathrm{Var}[\hat{f}_{\mathsf{OLH}(\epsilon)}(x) \cdot \ell] = n \cdot \frac{4\ell^2 \cdot e^\epsilon}{(e^\epsilon - 1)^2} \tag{5.2}$$

Comparing Equation (5.1) and Equation (5.2), when

$$d < \ell(4\ell - 1)e^\epsilon + 1, \tag{5.3}$$

using GRR itemset will lead to better accuracy. Note that by taking $\ell = 1$, Equation (5.3) is slightly different from the inequality of $d < 3e^\epsilon + 2$ from Chapter 3. This is because here we consider a more general setting where some user may have no item at all, while the setting

of Chapter 3 is that each user has exactly one item. We propose Adap, which becomes GRR or OLH adaptively (with new budget) based on Equation (5.3). That is,

$$\mathsf{Adap}(\epsilon) := \begin{cases} \mathsf{GRR}(\ln(\ell(e^\epsilon - 1) + 1) \text{ if } d < e^\epsilon \ell(4\ell - 1) + 1, \\ \mathsf{OLH}(\epsilon) \text{ otherwise.} \end{cases}$$

5.2.5 Choosing ℓ

To use PSFO, one needs to decide what value of ℓ to use. When ℓ is small, there is less variance but more bias (in the form of under estimation); when ℓ is large, there is more variance and less bias. To find the suitable ℓ, the high level idea is to find the right tradeoff between bias and variance.

When **identifying** candidate items, the goal is find the most frequent items (but not accurate frequencies for them), we propose to use a small ℓ. The intuition is that, while the bias is large when ℓ is small, the bias tends to be the same direction (namely under estimation) for all items. While the absolute values of the counts are very inaccurate, the relative order remain mostly unchanged. Note that it is possible the order is reversed after sampling (if one item appears more often in smaller transactions, and another item appears more often in larger transactions). To reduce this risk, we identify $2k$ candidate items (the optimal size of the candidate set is dependent on the data distribution; we tried different values and $2k$ appears to be a reasonable choice).

When **estimating** the actual frequency, one should use a larger ℓ to reduce bias. We propose to use the 90th percentile L of the length of the input itemsets. While under estimation can still occur, the degree is limited. Furthermore, when given the distribution of the lengths of input itemsets, we propose to correct this under estimation by multiplying the estimation by the factor:

$$u(L) = \frac{N}{N - \sum_{\ell=L+1}^{d} n^\ell(\ell - L)}. \tag{5.4}$$

Here N denotes the total number of items, n^ℓ denotes the number of users with itemset size ℓ, and $\sum_{\ell=L+1}^{d} N^\ell(\ell - L)$ gives the total number of missed items.

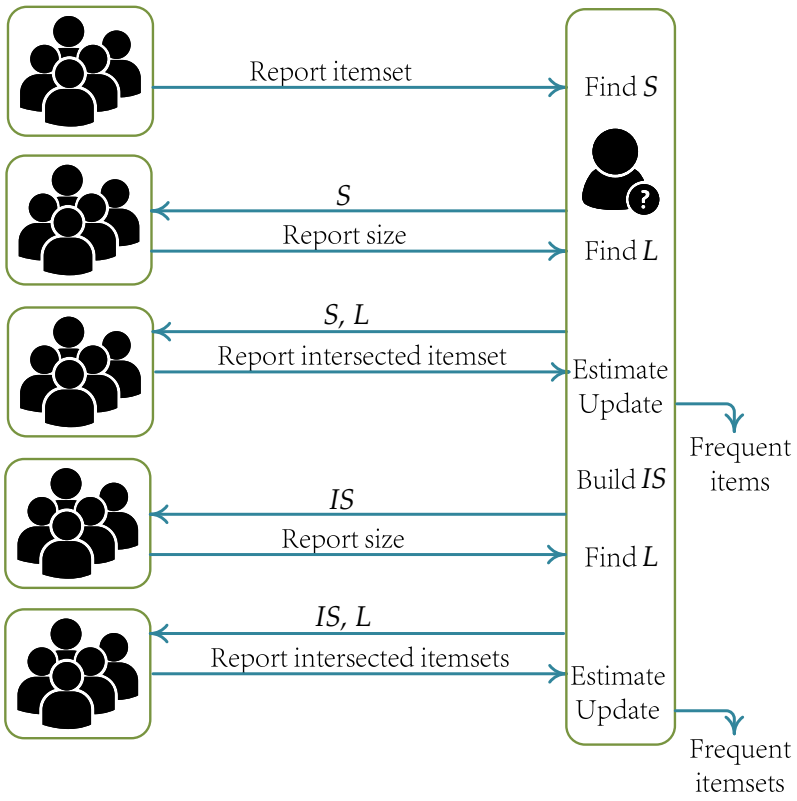


Figure 5.2. Illustration of SVIM and SVSM. The users to the left are partitioned into five groups. The aggregator to the right first runs SVIM with the first three groups, and find the frequent items. Then the aggregator interacts with the following two groups to find frequent itemsets.

5.3 Proposed Method

In this section, we propose solution for the frequent item and itemset mining. We first present Set-Value Item Mining (SVIM) protocol to find frequent items in the set-value setting. Based on the result from SVIM, we build Set-Value itemSet Mining (SVSM) protocol to find frequent itemsets. The high level protocol structure is given in Figure 5.2.

5.3.1 Frequent Item Mining

At a high level, SVIM works as follows: A set of candidate items are identified first. Then these items are estimated and updated. The users are partitioned into three groups,

each participating in a task. Given that each task requires privacy budget of ϵ, each user is protected by ϵ-LDP.

Step 1: Prune the Domain. When the domain is big (e.g., tens of thousands), the aggregator has to first narrow down the focus to a small candidate set. Specifically, in Step 1, each user reports with a randomly selected value from her private set with length limit set to 1:

$$\mathcal{A}_{\mathsf{PSFO}(1,\mathsf{Adap},\epsilon)}(\mathbf{v}).$$

The advantages of setting $\ell = 1$ are first, every user will report an item, making the signal strong; second, there is no extra cost of obtaining the exact L value.

The aggregator then estimates the frequency of the domain by

$$\hat{f}_{\mathsf{PSFO}(1,\mathsf{Adap},\epsilon)}(x),$$

and obtains the set S of the $2k$ most frequent items. S is then sent to users who participate in Step 2. Note that this phase is unnecessary when the original domain size close to or less than $2k$.

Step 2: Size Estimation. Having narrowed down the domain from I to S, the aggregator now estimates frequencies of items in S. As suggested by the analysis of PSFO (Section 5.2.5), the aggregator first finds the 90-th percentile L (in this step) and then uses it as the limit to estimate frequencies of S (next step).

To find L, each user in this task reports the size of the private set intersected with the candidate S, i.e.,

$$\mathcal{A}_{\mathsf{OLH}(\epsilon)}(|\mathbf{v} \cap S|).$$

There is no sampling involved in this step, because each user has only one value. Here OLH is as FO by default.

The aggregator in this step estimates the length distribution by calculating

$$\hat{f}_{\mathsf{OLH}(\epsilon)}(\ell)$$

for all $\ell \in [1, 2, \ldots, 2k]$, and finds the 90 percentile L. That is, the aggregator then finds the smallest L such that $\frac{\sum_{\ell=1}^{L} \hat{f}_{\mathsf{OLH}(\epsilon)}(\ell)}{\sum_{\ell=1}^{2k} \hat{f}_{\mathsf{OLH}(\epsilon)}(\ell)} > 0.9$. Information of S and L are then sent to the users for the next task.

Note that some of the estimates may be overwhelmed by noise, making it useless. For this reason, we use the significance threshold $T = \Phi^{-1}\left(1 - \frac{0.05}{2k}\right)\sqrt{\mathrm{Var}}$ to filter the estimates, where Φ^{-1} is the inverse of standard normal cumulative density function, and Var is specified by Equation (3.12). Specifically, the aggregator keeps estimates that are greater than T, and replaces all the others with zeros.

Step 3: Candidates Estimation. On receiving S and L, each of the rest of the users reports a value sampled from the intersection of his private set $\mathbf{v}$ and the candidate set S, padded to L, i.e.,

$$\mathcal{A}_{\mathsf{PSFO}(L,\mathsf{Adap},\epsilon)}(\mathbf{v} \cap S).$$

The aggregator can estimates the candidates by running

$$\hat{f}_{\mathsf{PSFO}(L,\mathsf{Adap},\epsilon)}(x),$$

for all $x \in S$. Since the 90-th percentile L is used as limit, the estimates are slightly under-estimate the truth. Therefore, the estimates are updated in the next step.

Step 4: Estimation Update. The update assumes that the missed count follow similar distribution as the reported ones. Given that L is the 90 percentile, the difference will not be significant. Thus the estimate for each item x is multiplied with a fixed update factor (the noisy version of Equation (5.4))

$$u'(L) \coloneqq \frac{\sum_{\ell=1}^{2k} \ell \cdot \hat{f}_{\mathsf{OLH}(\epsilon)}(\ell)}{\sum_{\ell=1}^{2k} \ell \cdot \hat{f}_{\mathsf{OLH}(\epsilon)}(\ell) - \sum_{\ell=L+1}^{2k} (\ell - L) \cdot \hat{f}_{\mathsf{OLH}(\epsilon)}(\ell)} \tag{5.5}$$

Note that there is no privacy concern in this step because no user is involved. The information is obtained from Step 2 and 3.

Difference from LDPMiner. The major differences between SVIM and LDPMiner are many. (1) In Phase 1 of SVIM, the limit is set to one, instead of the 90-th percentile of

lengths of full transactions. (2) In Phase 2 of SVIM, the limit is reduced from $|S|$ to the 90-th percentile L of the length of transactions when limited to items in S. (3) Phase 1 of LDPMiner uses the 90-th percentile; it was assumed that this is provided as input. In SVIM, the 90-th percentile of length is obtained in a way that satisfies LDP. (4) SVIM uses Adap instead of black box FO. (5) SVIM has an update step at the end, which uses the length distribution information to further reduces the bias. (6) In SVIM, users are partitioned into groups, each answering one separate question, instead of answering multiple questions each with part of ϵ. It is proved in Chapter 3.4.2 that this will make the overall result more accurate. (7) SVIM uses OLH, a more accurate FO introduced in Chapter 3. Since improvements (6) and (7) are not introduced in this paper, in the experiments, for a fair comparison, we evaluate on an improved version of LDPMiner. Specifically, OLH is used as the FO, and users are partitioned into groups. That is, the evaluation shows only differences due to (1), (2), (4), (5). Difference (3) means that SVIM is end-to-end private, and LDPMiner needs a data-dependent input.

5.3.2 Frequent Itemset Mining

The problem of mining frequent itemsets is similar to mining frequent items. The desired result becomes a set of itemsets instead of items. These frequent itemsets can be used, for example, by websites, to mine assocition rules and make recommendations. However, the task is much more challenging, because there are exponentially more itemsets to consider, and each user also has many more potential itemsets.

In this section, we introduce SVSM for finding frequent itemsets effectively. In the high level, the aggregator first obtains the frequent items by executing SVIM. The aggregator then constructs a candidate set of itemsets IS. Finally the set IS is estimated in a fashion similar to the latter part of SVIM.

Constructing Candidate Set. The challenging part of frequent itemset mining is to construct IS. There are exponentially many possible itemsets that can be frequent. If one can reduce it to a manageable range (thousands), one can cast the problem to the item

mining problem and run SVIM. Moreover, if size of IS is close to k, only the estimation of IS (latter part of SVIM) suffices.

Let S' be the k most frequent items returned by SVIM. To effectively further reduce the candidate size, we use information of the estimates of S'. Specifically, for an itemset $\mathbf{x}$, we first guess its frequency, denoted by $\tilde{f}_{\mathbf{x}}$, as the product of the estimates for all its items, i.e., $\tilde{f}_{\mathbf{x}} = \prod_{x \in \mathbf{x}} \hat{f}'(x)$, where $\hat{f}'(x) = \frac{0.9 \cdot \hat{f}(x)}{\max_{x \in S'} \hat{f}(x)}$ is the normalized estimate. The 0.9 factor of $\hat{f}'(x)$ serves to lower the normalized estimates for the most frequent item, because otherwise, the guessed frequency of any set without the most frequent item equals that of the set plus the most frequent item, which is unlikely to be true. Then $2k$ itemsets with highest guessing frequencies are selected to construct IS. The intuition is that, it is very unlikely that a frequent itemset is composed of several infrequent items (while it is theoretically possible). The guessing frequency is thus an effective measurement of the likelihood each itemset is among the frequent ones.

Formally, in SVSM, the domain IS is constructed as

$$IS \coloneqq \{\mathbf{x} : \mathbf{x} \subseteq S', 1 < |\mathbf{x}| < \log_2 k, \prod_{x \in \mathbf{x}} \hat{f}'(x) > t\},$$

where t is choosen so that $|IS| = 2k$.

Mining Frequent Itemset. After the domain IS is defined, the following protocol works similar to SVIM for frequent item mining. Note that step 1 is not necessary since IS is already small. For each user with value $\mathbf{v}$, a set of values from the domain IS is obtained first:

$$\mathbf{vs} = \{\mathbf{x} : \mathbf{x} \in IS, \mathbf{x} \subseteq \mathbf{v}\}$$

such that each itemset $\mathbf{x} \in \mathbf{vs}$ is a value in IS.

Then a group of users report the size of their $\mathbf{vs}$'s with FO:

$$\mathcal{A}_{\mathsf{OLH}(\epsilon)}(|\mathbf{vs}|).$$

After the aggregator evaluates the number of users that has ℓ itemsets for each $\ell \in [1, 2, \ldots, 2k]$, the aggregator finds the 90 percentile L and send it to the users in the final group, who then reports **vs** by

$$\mathcal{A}_{\mathsf{PSFO}(L,\mathsf{Adap},\epsilon)}(\mathbf{vs}).$$

The aggregator obtains the estimates by evaluating

$$\hat{f}_{\mathsf{PSFO}(L,\mathsf{Adap},\epsilon)}(\mathbf{x}) \cdot u'(L)$$

for any itemset $\mathbf{x} \in IS$, where $u'(L)$ is the update factor used for correcting bias (same format as Equation (5.5)), and get results for the heavy itemsets and their estimates. Note that as singletons are also sets, we also consider results obtained from SVIM when finding frequent itemsets.

5.4 Evaluation

Now we discuss experiments that evaluate different protocols. Basically, we want to answer the following questions: First, how many frequent items and itemsets can be effectively identified. Second, how much do our proposed protocols improve over existing ones.

As a highlight, in the POS dataset, our protocols can correctly identify around 45 frequent items (while existing ones can identify around 12), with much more accurate estimates (error is 3 orders of magnitudes less).

5.4.1 Experimental Setup

Environment. All algorithms are implemented in Python 2.7 and all the experiments are conducted on an Intel Core i7-4790 3.60GHz PC with 16GB memory. Each experiment is run 10 times, with mean and standard deviation reported.

Datasets. We run experiments on the following datasets:

- POS: A dataset containing merchant transactions of half a million users and 1657 categories.

- Kosarak: A dataset of click streams on a Hungarian website that contains around one million users and 42 thousand categories.
- Online: Similar to POS dataset, this is a dataset that contains merchant transactions of half a million users and 2603 categories.
- Synthesize: The dataset is generated by the IBM Synthetic Data Generation Code for Associations and Sequential Patterns 1.8 million transactions was generated, with 1000 categories. The average transaction size is 5.

For brevity, we only plot results for the one dataset (POS). The detailed results for other datasets are deferred to the supplementary section.

Metrics. To measure utility, we use the following metrics. Define x_i as the i-th most frequent value (x_i is an item in the task of item mining and an itemset in itemset mining). Let the ground truth for top k values as $\mathbf{x}_t = \{x_1, x_2, \ldots, x_k\}$. Denote the k values identified by the protocol using $\mathbf{x}_r$. Then $\mathbf{x}_t \cap \mathbf{x}_r$ is the set of real top-k values that are identified by the protocol.

1. Normalized Cumulative Rank (NCR). For each value x, we assign a quality function $q(\cdot)$ to each value, and use the Normalized Cumulative Gain (NCG) metric:

$$\text{NCG} = \frac{\sum_{x \in \mathbf{x}_r} q(x)}{\sum_{x \in \mathbf{x}_t} q(x)}.$$

We instantiate the quality function using x's rank as follows: the highest ranked value has a score of k (i.e., $q(x_1) = k$), the next one has score $k-1$, and so on; the k-th value has a score of 1, and all other values have scores of 0. To normalize this into a value between 0 and 1, we divide the sum of scores by the maximum possible score, i.e., $\frac{k(k+1)}{2}$. This gives rise to what we call the Normalized Cumulative Rank (NCR); this metric uses the true rank information of the top-k values.

2. Mean Squared Error (Var): We measure the estimation accuracy by squared errors. That is,

$$\frac{1}{|\mathbf{x}_t \cap \mathbf{x}_r|} \sum_{x \in \mathbf{x}_t \cap \mathbf{x}_r} \left(f_x \cdot n - \hat{f}(x) \right)^2,$$

As the quantity converges to the theoretical variance, we also use Var to denote it. Note that we only account heavy hitters that are successfully identified by the protocol, i.e., $x \in \mathbf{x}_t \cap \mathbf{x}_r$.

5.4.2 Evaluation of Item Mining

For the item mining problem, our main focus is to compare the performance of our proposed method SVIM, and the existing method, LDPMiner. We implemented them as follows:

LDPMiner is almost implemented as described in [21]. For a fair comparison, we made two modifications. First, we partition the users into two groups. The first group focus on finding S, while the second focus on estimating S. Users in each group use the full privacy budget ϵ to report. It is proven in Chapter 3.4.2 that by this way, the overall utility is better, compared to keeping asking all the users multiple questions, with splited privacy budget. Second, to get the 90th percentile L, an additional group of users are assigned to report the size of their private set. As a result, there will be three groups, 10% of users report size in advance, 40% report in the first phase, and 50% report in the second phase.

For SVIM, we do the similar thing. Half of the users report based on the original itemsets to find the candidate set S, and the other half report after seeing the candidate set to estimate S. The difference is, the 90th percentile L is used when estimating S. Therefore, 10% of all users are allocated to estimate L from the second half. That is, 50% report in the first phase, 10% of users report size of the their itemsets intersected with S, and 40% report one actual item.

To demonstrate the precise effect of each design detail, we also line up several intermediate protocols between LDPMiner and SVIM. We present them with synonyms (that specify the FO and ℓ used in both tasks) to highlight the difference as follows:

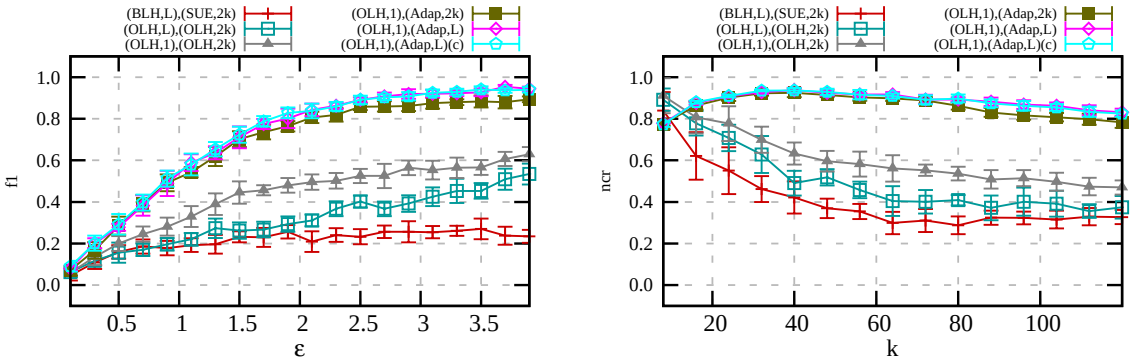
- $(\mathsf{BLH}, L), (\mathsf{SUE}, 2k)$: LDPMiner. LDPMiner uses two FO's BLH [8] and SUE [2]. It is proven in Chapter 3 that the two performs not as good as OLH.
- $(\mathsf{OLH}, L), (\mathsf{OLH}, 2k)$: The frequency oracles are replaced with OLH.
- $(\mathsf{OLH}, 1), (\mathsf{OLH}, 2k)$: The first phase uses $\ell = 1$. Note that L is no longer needed, so there are two groups each consists of half of the users.
- $(\mathsf{OLH}, 1), (\mathsf{Adap}, 2k)$: The second phase uses adaptive frequency oracle.
- $(\mathsf{OLH}, 1), (\mathsf{Adap}, L)$: The second phase uses L. An extra group of 10% of users are assigned to estimate that.
- $(\mathsf{OLH}, 1), (\mathsf{Adap}, L)(c)$: The final results are updated based on the length distribution. This is the SVIM.

Note that the allocation of 10% of users for length distribution is not fully justified. This is because the optimal allocation depends heavily on the dataset, and 10% seems a reasonable choice.

Detailed Results. In Figure 5.3, we evaluate the above six protocols on POS dataset, and plot the NCR and Var scores. Overall, the identification accuracy (indicated by NCR) increases with ϵ, and decreases with k. Similarly, the estimation accuracy becomes better (as the indicator Var decreases) when ϵ is larger, and worse (Var increases) if k is larger. Now we analyze performance of each competitor in more detail.

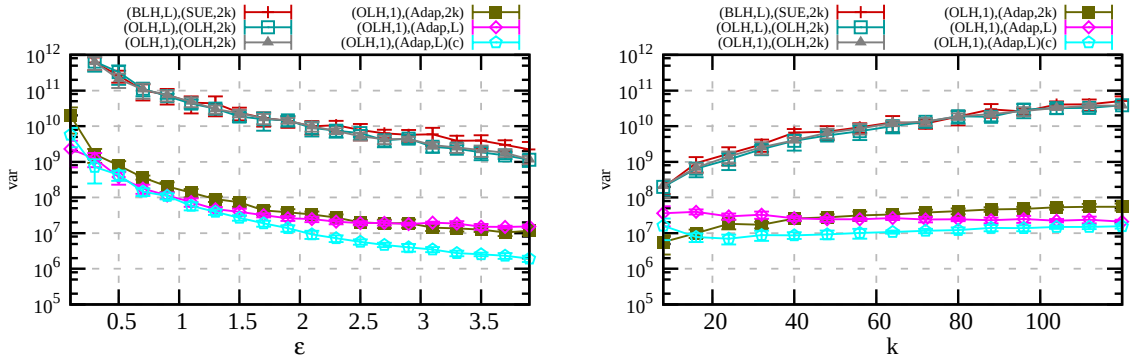
1. $(\mathsf{BLH}, L), (\mathsf{SUE}, 2k) \to (\mathsf{OLH}, L), (\mathsf{OLH}, 2k)$: First of all, we observe the identification accuracy improves when the FO in the first phase is changed from BLH to OLH. This happens because, by using OLH, a more accurate S will be returned, and by using OLH in the second phase, one can better identify the top k items. Note that the estimation accuracy actually does not improve significantly, because better FO does a better job at reducing the noise for the lower ranked values (thus NCR is higher). The estimation improvement is nearly unnoticeable in the log based figures.

2. $(\mathsf{OLH}, L), (\mathsf{OLH}, 2k) \to (\mathsf{OLH}, 1), (\mathsf{OLH}, 2k)$: One major NCR improvement happens when the length limit is changed from the 90th percentile L to 1. To this point, the top $2k$



(a) POS, NCR, vary $\epsilon, k = 64$

(b) POS, NCR vary $k, \epsilon = 2$



(c) POS, Var, vary $\epsilon, k = 64$

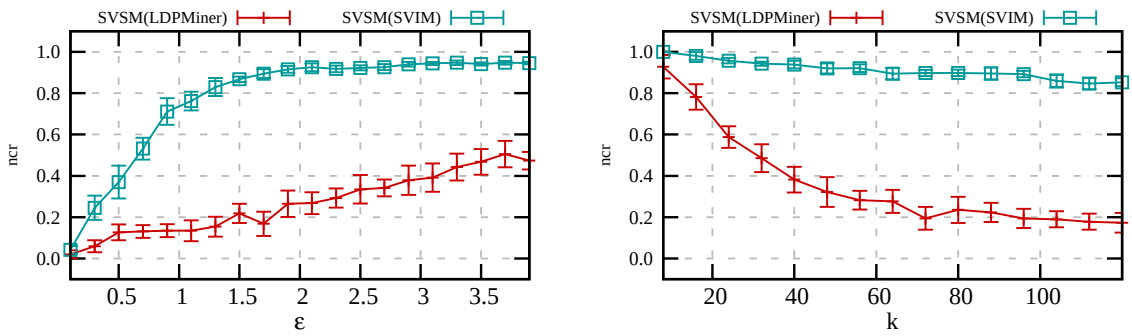
(d) POS, Var vary $k, \epsilon = 2$

Figure 5.3. Singleton identification.

items returned by the first phase contains most of the top k items. The NCR bottle neck lies on the second phase, which cannot effectively identify the top k from the $2k$ items. Note that the estimation accuracy does not improve because the same FO is used in the second phase.

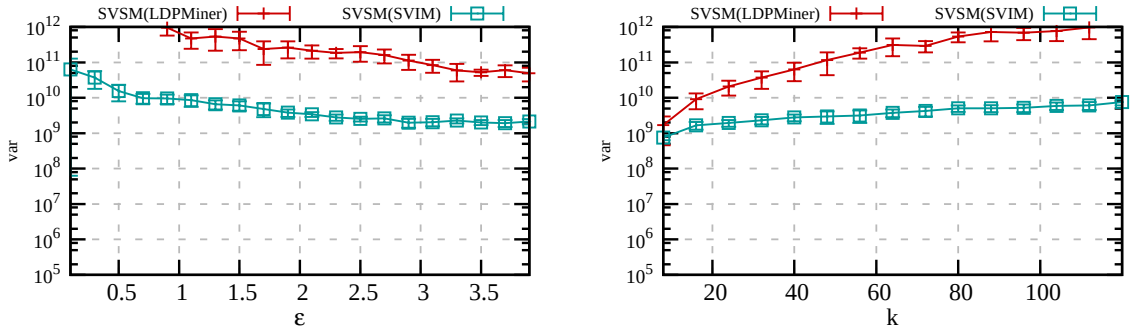
3. $(\mathsf{OLH}, 1), (\mathsf{OLH}, 2k) \rightarrow (\mathsf{OLH}, 1), (\mathsf{Adap}, 2k)$: The most significant improvement happens when changing from OLH to Adap in the second phase. Both identification and estimation accuracy significantly (NCR almost doubled, and Var reduced by two magnitudes). This is because Adap significantly reduces the variance (from a factor of $(2k)^2$ to $2k$).

4. $(\mathsf{OLH}, 1), (\mathsf{Adap}, L)$ and $(\mathsf{OLH}, 1), (\mathsf{Adap}, L)(c)$: By reducing $2k$ to the 90th percentile L in the second phase, the results are further improved. Note that the improvement is not



(a) NCR, vary $\epsilon, k = 64$

(b) NCR vary $k, \epsilon = 2$



(c) Var, vary $\epsilon, k = 64$

(d) Var, vary $k, \epsilon = 2$

Figure 5.4. POS Itemset Mining Results.

that significant but still meaningful. This is partly because an additional 10% of users are assigned to estimate the size distribution (to find L and update the results).

Remark. Because of the noisy nature (noise is in the order of $\sqrt{n}$) of the local setting of DP, in order to get meaningful information, one has to increase ϵ or n (or both). When the number of users is not sufficiently large, as in our experiment, the improvement is not significant in the small ϵ range, as being used by experiments of centralized DP (e.g., 0.1). However, in the case of deployed LDP protocol (Google uses $\epsilon > 4$ [2], and Apple uses $\epsilon = 1$ or 2 [23]), the advantage of the proposed protocol is profound.

5.4.3 Evaluation of Itemset Mining

We evaluate the effectiveness of SVSM. We want to answer the questions how many itemsets can be identified, and the effectiveness of using SVIM in SVSM.

We implement SVSM as follows, half of the users are allocated to find frequent items first. Then the set IS is constructed and estimated, by taking each of the element of it as an independent item. To compare the effect of SVIM over LDPMiner, we also instantiate SVSM using LDPMiner. Specifically, half of the users are allocated to find frequent items using LDPMiner; then IS is constructed similarly; finally, Phase 2 of LDPMiner is executed to estimate frequency of IS and output the most frequent k itemsets. Note that the $50\% - 50\%$ allocation is used since mining singletons and itemsets are two goals. One can allocate more users to singletons if singleton mining is more important.

Detailed Results. Figure 5.4 shows the results of mining frequent itemsets. As we can see from the upper two sub-figures, when fixing $k = 64$, the proposed SVSM protocol (instantiated with SVIM, as default) can achieve the NCR score of 0.7 at $\epsilon = 1$ and 0.9 when $\epsilon = 2$. As to when LDPMiner is used to instantiate SVSM, the utility drops to around 0.2. When ϵ is fixed at 2, the improvement of SVIM over LDPMiner is also significant, especially when k is greater than 64 (SVSM-SVIM keeps NCR greater than 0.8, while NCR for SVSM-LDPMiner drops to below 0.2). This suggests that SVSM with LDPMiner can effectively find only around 10 most frequent itemsets, while SVSM with SVIM can find around 70, demonstrating a $7\times$ improvement.

For the estimation accuracy shown by the bottom two sub-figures, we can see that the estimation error drops with ϵ, and increases with k. When using LDPMiner in SVSM the error is two magnitudes greater than using SVIM. This effect is more significant when k is greater than 64. This is because Var for LDPMiner is heavily dependent on k, while SVIM not.

5.5 Supplementary Results

5.5.1 (ϵ, δ)-LDP and Limited Amplification Effect

In (ϵ, δ)-LDP, the value δ (which is typically very small) has an intuitive interpretation of "failure" probability. That is, with probability $1 - \delta$, $\mathcal{A}$ is ϵ-LDP. When $\delta = 0$, $(\epsilon, 0)$-LDP becomes ϵ-LDP.

Definition 5.5.1 ((ϵ, δ) Local Differential Privacy)**.** *An algorithm $\mathcal{A}$ satisfies (ϵ, δ)-local differential privacy ((ϵ, δ)-LDP), where $\epsilon \geq 0$, and $0 \leq \delta < 1$ if and only if for any input $\boldsymbol{v}_1, \boldsymbol{v}_2 \subseteq I$, we have*

$$\forall T \subseteq Range(\mathcal{A}) : \ \mathsf{Pr}\left[\mathcal{A}(\boldsymbol{v}_1) \in T\right] \leq e^{\epsilon} \mathsf{Pr}\left[\mathcal{A}(\boldsymbol{v}_2) \in T\right] + \delta,$$

where $Range(\mathcal{A})$ denotes the set of all possible outputs of the algorithm $\mathcal{A}$.

To apply the privacy amplification, one uses δ to measure the probability that failure (multiple values are hashed to the same value) happens, and derive the corresponding ϵ' that OLH can use. For example, when $\ell = 2$, the probability both the user's items are hashed into the same value by the chosen hash function is $\delta = \frac{1}{g}$, where $g = \lceil e^{\epsilon'} + 1 \rceil$ is the range of the hash function. Under the condition the user's items are hashed to at least two results, OLH can be used with $\epsilon' = \ln(2e^{\epsilon} - 1)$.

Theorem 5.5.1 ((ϵ, δ)-LDP by $\mathsf{OLH}(\epsilon')$)**.** *$\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_{\ell}(\cdot))$ satisfies (ϵ, δ)-LDP, where $\epsilon' = \ln\left(\frac{\ell}{\ell'} \cdot (e^{\epsilon} - 1) + 1\right)$, and ℓ' is an integer such that*

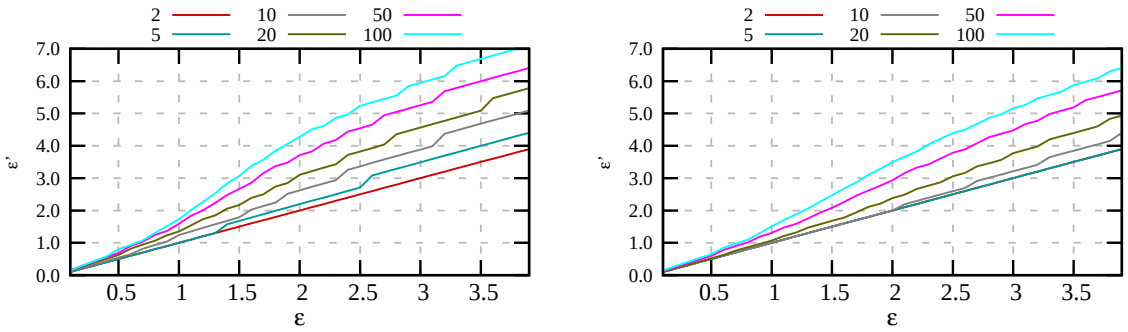
$$\binom{\ell}{\ell' + 1} \cdot \frac{1}{\lceil \frac{\ell}{\ell'} \cdot (e^{\epsilon} - 1) + 2 \rceil^{\ell'}} \leq \delta.$$

That is, for any $\epsilon \geq 0$, any input $\boldsymbol{v}_1, \boldsymbol{v}_2 \subseteq I$, any set of output $T \subseteq Range(\mathcal{A}_{\mathsf{PSFO}(\ell, \mathsf{GRR}, \epsilon')})$,

$$\begin{aligned} &\mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}(\boldsymbol{v}_1, \ell)) \in T\right] \\ \leq\ & e^{\epsilon} \cdot \mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}(\boldsymbol{v}_2, \ell)) \in T\right] + \delta. \end{aligned} \tag{5.6}$$

Proof: To prove (5.6), it is equivalent to first prove that a "failure" event, where more than ℓ' items in $\mathbf{v}_1$ are hashed to the same value, happens with probability less than δ, and then prove that under the condition the "failure" event does not happen, $\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_{\ell}(\cdot))$ satisfies ϵ-LDP.

Given that the hash function is chosen randomly, and the hash family is random, bounding the "failure" probability is equivalent to bounding the probability of throwing ℓ balls



(a) Amplification of OLH with $\delta = 10^{-3}$ (b) Amplification of OLH with $\delta = 10^{-9}$

Figure 5.5. Privacy amplification effect for different ℓ.

randomly into g bins, and the max load is more than ℓ'. The probability can be calculated as follows:

Let $E_{i,a}$ be the event that bin i contains more than a balls, then

$$\mathsf{Pr}\left[E_{i,a}\right] = \binom{\ell}{a+1}\frac{1}{g^{a+1}}$$

By union bound, we know that

$$\delta = \mathsf{Pr}\left[\bigcup_{i\in[g]} E_{i,\ell'}\right] \leq \sum_i \mathsf{Pr}\left[E_{i,\ell'}\right] = \binom{\ell}{\ell'+1}\frac{1}{g^{\ell'}}$$

where $g = \lceil e^{\epsilon'} + 1\rceil = \lceil \frac{\ell}{\ell'} \cdot (e^{\epsilon} - 1) + 2\rceil$.

Now it suffices to prove that for any $\epsilon \geq 0$, any $\mathbf{v}_1, \mathbf{v}_2 \subseteq I$, any possible hash function H (such that at most ℓ' items are hashed into the same value), and any $t \in [g]$, $\frac{p_1}{p_2} \leq e^{\epsilon}$, where

$$p_1 = \mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_\ell(\mathbf{v}_1)) = \langle H, t\rangle\right], \text{ and}$$
$$p_2 = \mathsf{Pr}\left[\mathcal{A}_{\mathsf{OLH}(\epsilon')}(\mathsf{PS}_\ell(\mathbf{v}_2)) = \langle H, t\rangle\right].$$

We first upper bound p_1,

$$\begin{aligned}
p_1 =& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon)}(H(\mathsf{PS}_\ell(\mathbf{v}_1))) = t | H\right] \\
=& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \Big(\mathsf{Pr}\left[v \text{ is sampled } \wedge H(v) = t\right] p' \\
&+ \mathsf{Pr}\left[v \text{ is sampled } \wedge H(v) \neq t\right] q' \Big) \\
\leq& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \Bigg(\frac{\ell'}{\max\{|\mathbf{v}_1|, \ell\}} \cdot p' \\
&+ \frac{\max\{|\mathbf{v}_1|, \ell\} - \ell'}{\max\{|\mathbf{v}_1|, \ell\}} \cdot q' \Bigg)
\end{aligned}$$

The equality holds when $H(v) = t$ for all $\mathbf{v}_1$. Similarly, we lower bound p_2,

$$\begin{aligned}
p_2 =& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \mathsf{Pr}\left[\mathcal{A}_{\mathsf{GRR}(\epsilon)}(H(\mathsf{PS}_\ell(\mathbf{v}_2))) = t | H\right] \\
=& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \Big(\mathsf{Pr}\left[v \text{ is sampled } \wedge H(v) = t\right] p' \\
&+ \mathsf{Pr}\left[v \text{ is sampled } \wedge H(v) \neq t\right] q' \Big) \\
\geq& \mathsf{Pr}\left[H \text{ is picked}\right] \cdot \Bigg(\frac{0}{\max\{|\mathbf{v}_1|, \ell\}} \cdot p' \\
&+ \frac{\max\{|\mathbf{v}_1|, \ell\}}{\max\{|\mathbf{v}_1|, \ell\}} \cdot q' \Bigg) = \mathsf{Pr}\left[H \text{ is picked}\right] \cdot q'
\end{aligned}$$

The equality holds when none of the items from $\mathbf{v}_2$ are hashed to t by H. Thus, we now bound $\frac{p_1}{p_2}$:

$$\begin{aligned}
\frac{p_1}{p_2} &\leq \frac{p'}{q'} \cdot \frac{\ell'}{\max\{|\mathbf{v}_1|, \ell\}} + \frac{\max\{|\mathbf{v}_1|, \ell\} - \ell'}{\max\{|\mathbf{v}_1|, \ell\}} \\
&= 1 + \frac{\ell'}{\max\{|\mathbf{v}_1|, \ell\}} \cdot \left(\frac{p'}{q'} - 1 \right) \\
&\leq 1 + \frac{\ell'}{\ell} \cdot \left(e^{\epsilon'} - 1 \right) \\
&= 1 + \frac{\ell'}{\ell} \cdot \left(\frac{\ell}{\ell'} \cdot (e^{\epsilon} - 1) + 1 - 1 \right) = e^{\epsilon}.
\end{aligned}$$

The equality is achieved when $H(v) = t$ for all $\mathbf{v}_1$ while $H(v) \neq t$ for all $\mathbf{v}_2$.

Table 5.2. Numerical value of ϵ' under different ϵ and ℓ. The upper part is for $\delta = 10^{-3}$, and the lower part is for $\delta = 10^{-9}$.

ϵ \ ℓ	2	5	10	20	50	100
0.1	0.10	0.10	0.11	0.13	0.15	0.15
0.5	0.50	0.50	0.54	0.62	0.68	0.80
1.0	1.00	1.00	1.24	1.35	1.59	1.73
2.0	2.00	2.20	2.62	3.10	3.71	4.28
4.0	4.00	4.50	5.19	5.88	6.80	7.20
0.1	0.10	0.10	0.10	0.10	0.12	0.14
0.5	0.50	0.50	0.50	0.52	0.59	0.65
1.0	1.00	1.00	1.00	1.07	1.30	1.51
2.0	2.00	2.00	2.00	2.38	2.93	3.49
4.0	4.00	4.00	4.50	5.04	5.82	6.51

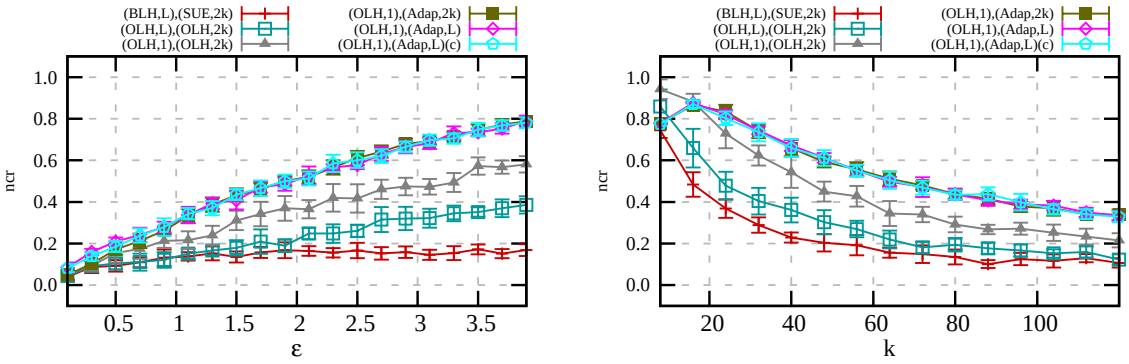
□

The theorem above gives us the formula to calculate δ and ϵ' for any ℓ'. Therefore, if δ is specified, we are able to come up with the highest ϵ'. Table 5.2 and Figure 5.5 give results of ϵ' given ϵ and ℓ, under the condition δ equals 10^{-3} and 10^{-9}, respectively. We can see $\epsilon' \geq \epsilon$, the difference becomes more significant when ϵ or ℓ is large. However, the increased amount is less than that for GRR, as shown in Table 5.1 and Figure 5.1.

Note that however, the (ϵ, δ)-LDP notion is strictly weaker (less secure) than ϵ-LDP and thus not directly comparable here.

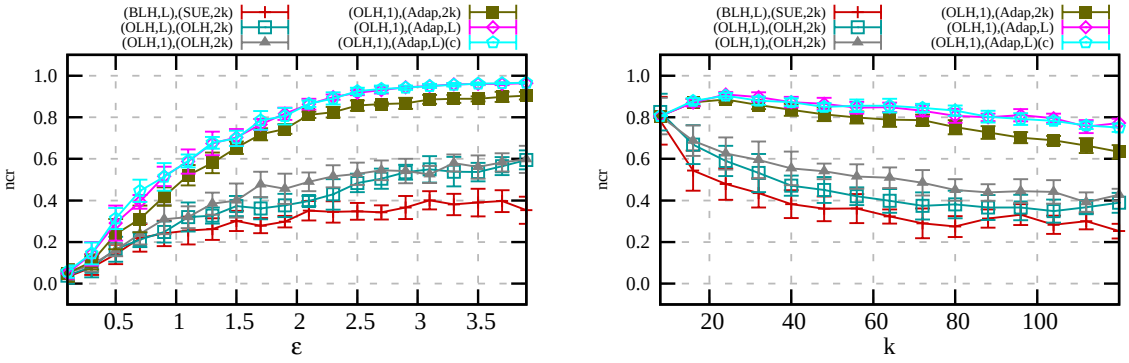
5.5.2 Additional Results

Item Mining. We report experimental results of item mining for the datasets of Kosarak, Online and Synthesize in Figures 5.6 and 5.7. We can see similar trends as that of Figure 5.3. Note that performance on different dataset is slightly different, because of different size, distribution, etc. Specifically, NCR and Var are worse in the Kosarak dataset, than that on the others, because the original domain is big (42 thousand, while the others are 1



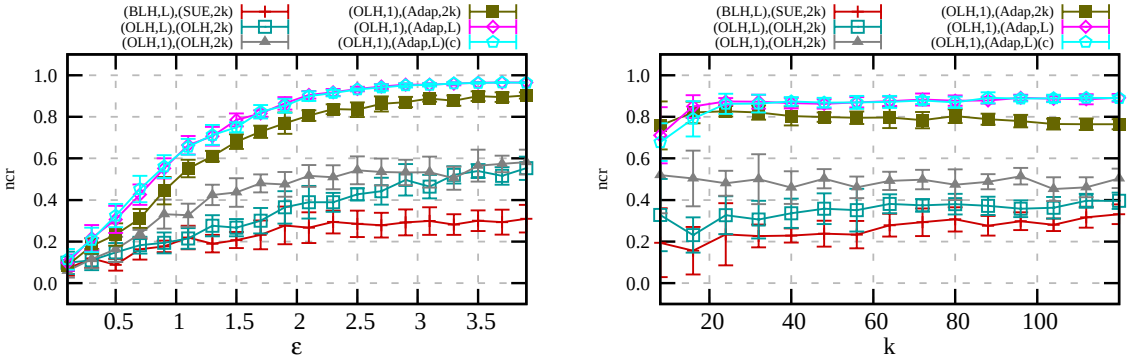
(a) Kosarak, NCR, vary $\epsilon, k = 64$

(b) Kosarak, NCR vary $k, \epsilon = 2$



(c) Online, NCR, vary $\epsilon, k = 64$

(d) Online, NCR vary $k, \epsilon = 2$



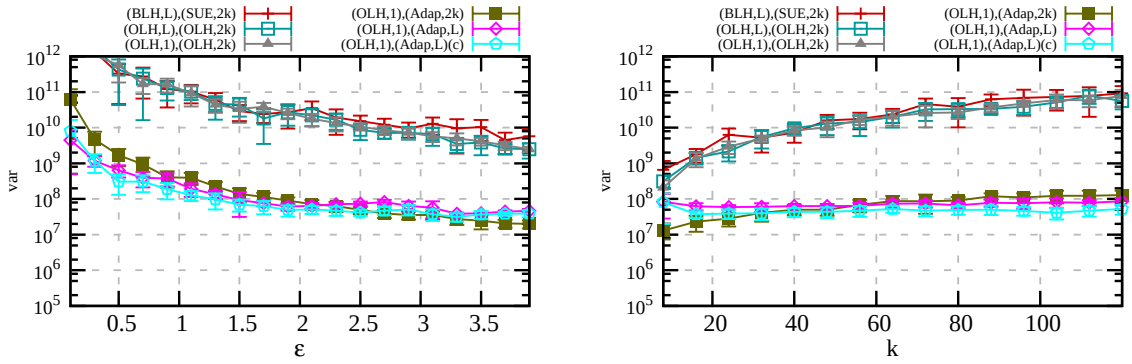
(e) Synthetic, NCR, vary $\epsilon, k = 64$

(f) Synthetic, NCR vary $k, \epsilon = 2$

Figure 5.6. More results on singleton identification.

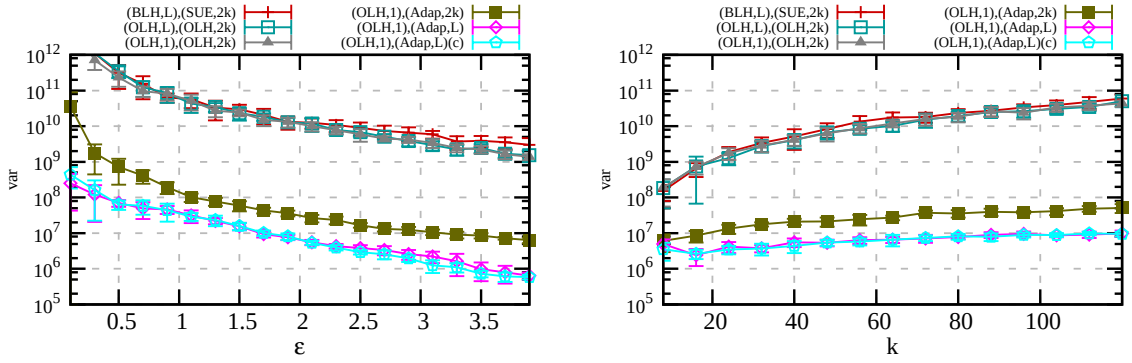
to 3 thousand). Overall, the proposed method SVIM works persistently better than its competitors.

Itemset Mining. We also plot results for itemset mining in Figure 5.8. Results for the synthetic dataset is not included because there is no frequent itemset (the items from the



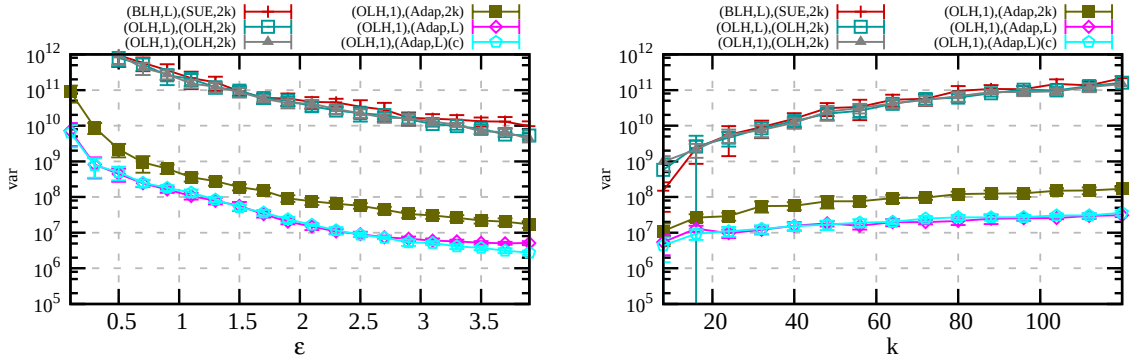
(a) Kosarak, Var, vary $\epsilon, k = 64$

(b) Kosarak, Var vary $k, \epsilon = 2$



(c) Online, Var, vary $\epsilon, k = 64$

(d) Online, Var vary $k, \epsilon = 2$

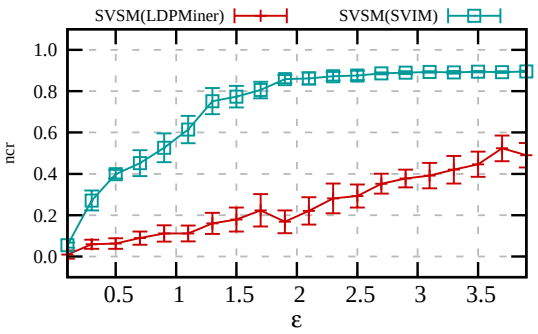


(e) Synthetic, Var, vary $\epsilon, k = 64$

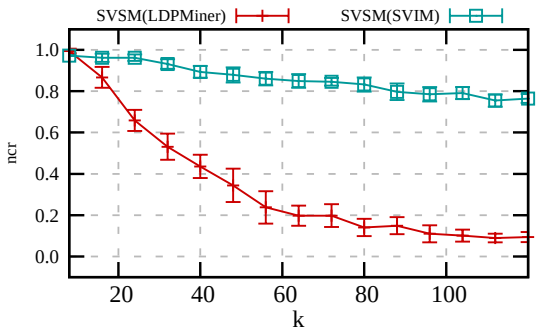
(f) Synthetic, Var vary $k, \epsilon = 2$

Figure 5.7. More results on singleton estimation.

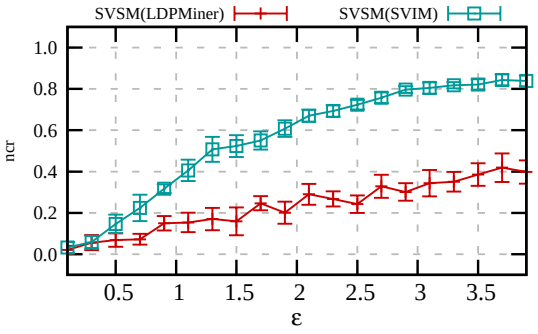
generator are independent). For the others, we can still see similar trends and that our proposed solution works persistently better.



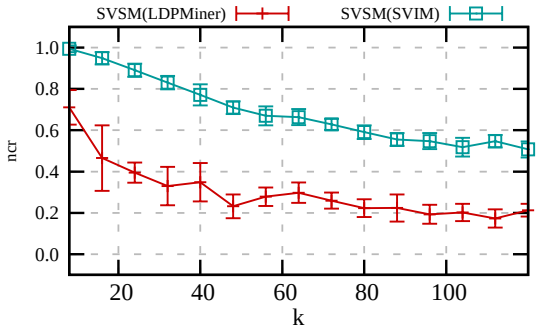
(a) Kosarak NCR, vary $\epsilon, k = 64$



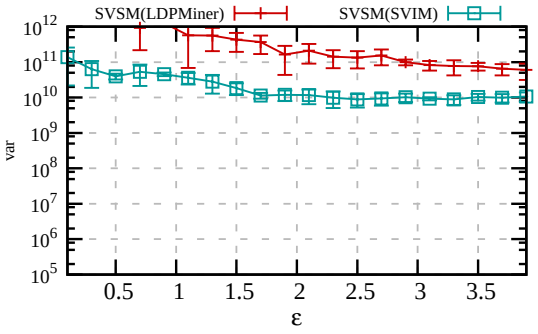
(b) Kosarak NCR vary $k, \epsilon = 2$



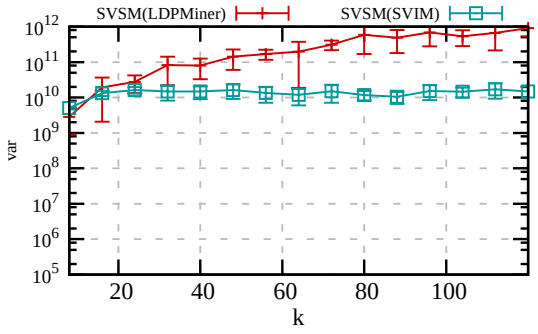
(c) Online NCR, vary $\epsilon, k = 64$



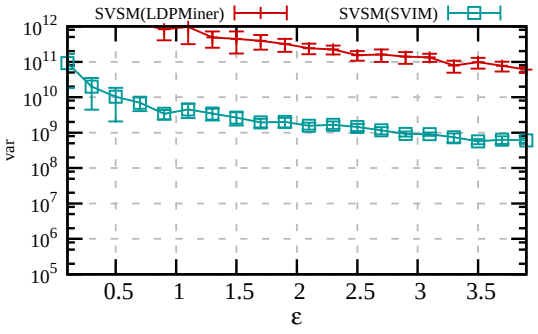
(d) Online NCR vary $k, \epsilon = 2$



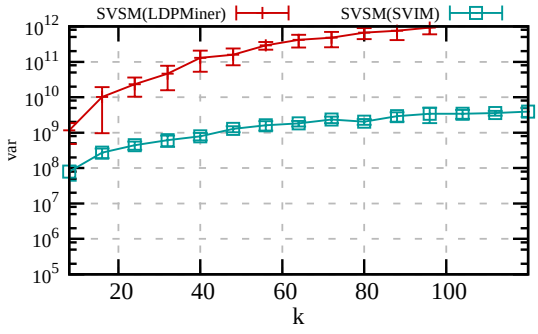
(e) Kosarak Var, vary $\epsilon, k = 64$



(f) Kosarak Var, vary $k, \epsilon = 2$



(g) Online Var, vary $\epsilon, k = 64$



(h) Online Var, vary $k, \epsilon = 2$

Figure 5.8. More results on itemset mining results for Kosarak dataset.

6. MARGINAL RELEASE

(A version of this chapter has been previously published in ACM CCS 2018 [24].)

Marginal tables are the workhorse of capturing the correlations among a set of attributes. Many analysis tasks require the availability of marginal statistics on multidimensional datasets. For example, finding correlations or fitting sophisticated prediction models.

We consider the problem of constructing marginal tables given a set of user's multi-dimensional data while satisfying local differential privacy (LDP). In this case, the aggregator is interested in marginal tables over some subsets of attributes.

6.1 Problem Definition and Existing Solutions

In the marginal release problem, we consider the setting where each user has multiple attributes, and the aggregator is interested in the joint distribution of some attributes. Such multi-dimension settings occur frequently in the situation where LDP is applied. In [25], [26], researchers studied the problem of constructing marginals in the LDP setting.

6.1.1 Problem Definition: Centralized Setting

We assume that there are m attributes $\mathbb{A} = \{a_1, a_2, \ldots, a_m\}$. Each attribute a_i has d_i possible values. Wlog, we assume that the values for a_i are $[d_i] \coloneqq \{0, 1, \cdots, d_i - 1\}$. Each user has one value for each attribute. Thus user j's value is a m-dimensional vector, denoted by $v^j = \langle v_1^j, v_2^j, \ldots, v_m^j \rangle$ such that $v_i^j \in [d_i]$ for each i. The full domain for the users' values is given by $D = [d_1] \times [d_2] \times \cdots \times [d_d]$, in which $\times$ denotes Cartesian product. The domain D has size $|D| = \prod_{i=1}^{m} d_i$.

Let us first consider the setting of answering marginal queries in the *centralized setting*, where the server has all users' data. For a population of n users, the *full contingency table* gives, for each value $v \in D$, the fraction of users having the value v. We use F to denote the full contingency table, and call the fraction for each value $v \in D$ a cell in the full contingency table.

	Gender	Age
v^1	male	teenager
v^2	female	teenager
v^3	female	adult
v^4	female	adult
$\cdots$	$\cdots$	$\cdots$
v^n	male	elderly

(a) Dataset.

v	$\mathsf{F}(v)$
⟨male, teenager⟩	0.20
⟨male, adult⟩	0.15
⟨male, elderly⟩	0.20
⟨female, teenager⟩	0.15
⟨female, adult⟩	0.20
⟨female, elderly⟩	0.10

(b) Full contingency table.

v	$\mathsf{M}_{\{\text{gender}\}}(v)$
⟨male,∗⟩	0.55
⟨female,∗⟩	0.45

(c) Marginal table for gender.

v	$\mathsf{M}_{\{\text{age}\}}(v)$
⟨∗,teenager⟩	0.35
⟨∗,adult ⟩	0.35
⟨∗,elderly⟩	0.30

(d) Marginal table for age.

Figure 6.1. Example of the dataset, the full contingency table, and the marginal tables.

The full contingency table gives the joint distribution of all attributes in $\mathbb{A}$. However, when the domain size is very large, e.g., when there are many attributes, computing the full contingency table can be prohibitively expensive. Oftentimes, one is interested in the joint distribution of some subsets of the attributes. Given a set of attributes $A \subseteq \mathbb{A}$, we use $V_A = \{\langle v_1, v_2, \ldots, v_m\rangle : v_i \in [d_i] \text{ if } a_i \in A, \text{otherwise } v_i = *\}$ to denote the set of all possible values specified by A.

When given a set A of k attributes, the *k-way marginal over A*, denoted by M_A, gives the fraction of users having each value in V_A. We call the fraction for each value $v \in V_A$ a cell of the marginal table. M_A contains fewer cells than the full contingency table F. Each cell in M_A can be computed from summing over the values in the cells in F that have the same values on the attributes in A.

Figure 6.1 gives an example where each user has two attributes gender and age. In the centralized setting, the server has access to the raw dataset Figure 6.1a, from which, it can compute the full contingency table (Figure 6.1b). The two marginal tables (Figure 6.1c, and Figure 6.1d) can be computed from the contingency table.

6.1.2 Problem Definition: Local Setting

In the local setting, the aggregator does not have access to the raw dataset, such as the one shown in Figure 6.1a. Instead, each user possesses one row of the dataset and sends a randomly perturbed value based on it. Our goal is to have the aggregator to use the perturbed reports to compute with reasonable accuracy any k-way marginal. Some methods (such as those proposed in [25]) require a specification of the maximum k ahead of time. Our proposed method can support queries of arbitrary k values.

To measure the utility empirically, we use *sum of squared error (SSE)*, i.e., the square of the L_2 distance between the true marginal M_A and the reconstructed T_A. When we query many k-way marginals, we calculate the SSE for each marginal, and use the average SSE as the indicator of a method's accuracy.

The reconstructed T_A can be viewed as a random variable since random noises are added in the process to satisfy LDP. When a method is able to produce an unbiased estimation, the expected value of T_A is the true marginal M_A, and the expected value of SSE is the variance of the random variable T_A.

Figure 6.1 gives an example where each user has two attributes gender and age. The goal is to construct all the marginal tables. Each user's private value corresponds to a row in Figure 6.1a. No one has the full view of the whole dataset. To construct the marginal tables Figure 6.1c and Figure 6.1d, one can let each user report the two values (using an FO as described earlier), aggregate the users' reports to construct the full contingency table (with some noise), and build the marginal tables. This method is more formally described in the following.

See Table 6.1 for the list of notations.

6.1.3 Full Contingency Table Method (FC)

To estimate M, one straightforward approach is to estimate the full contingency table F first, and then construct M from F. We call this approach the Full Contingency (FC) table method. In this method, each user reports her value $v \in D$ using an FO protocol. The

Table 6.1. List of Notations

Symbol	Description
n	The total number of users
v^j	Value of user j
m	Number of attributes
$\mathbb{A}$	The set of all attributes
a_i	Attribute i
d_i	Number of possible values for attribute a_i
F	The full contingency table
A	Some set of attributes
M_A	The marginal table of attribute set A
t	Number of marginal tables output by our method
s	Size of each marginal table in our method

aggregator estimates the frequency of each value in the full domain. Once having the full contingency table, the aggregator can compute any k-way marginal.

The main shortcoming of FC is that, since one has to query the frequency of each value in the full domain of all attributes, the time complexity and space complexity grows exponentially with the number of attribute m and can be prohibitively expensive.

Furthermore, even when it is feasible to construct the full contingency table, computing marginals from a noisy full contingency table can have high variance. For example, suppose we have 32 binary attributes, the domain size is thus 2^{32}. When constructing a 4-way marginal, each value in the 4-way marginal is the result of summing up 2^{28} noisy entries in the full contingency table. Let Var_0 be the variance of estimating each single cell in the full contingency table, the variance of each cell in the reconstructed marginal is then $2^{28} \times \mathrm{Var}_0$, and the expected SSE is $2^4 \times 2^{28} \times \mathrm{Var}_0 = 2^{32} \times \mathrm{Var}_0$. In general, the variance of computing k-way marginals from the noisy full contingency table is

$$\mathrm{Var}_{\mathsf{FC}} = 2^m \cdot \mathrm{Var}_0 \tag{6.1}$$

6.1.4 All Marginal Method (AM)

To mitigate the exponential dependency on m, one can construct all the k-way marginals directly. There are two alternatives, one is to divide the privacy budget ϵ into $\binom{m}{k}$ pieces, and have each user reports $\binom{m}{k}$ times, once for each k-way marginal. The second is to divide the user population into $\binom{m}{k}$ disjoint groups, and have users in each group report one k-way marginal. Under the LDP setting, it is generally better to divide the population than dividing the privacy budget (Chapter 3.4.2).

Under LDP, estimating fraction frequencies is less accurate with a smaller group than with a larger group, because the noises have larger impact when the true counts are small. The variance is inversely proportional to the group size. Thus dividing the population into $\binom{m}{k}$ groups will add a $\binom{m}{k}$ factor to the variance. This factor results in the following variance.

$$\mathrm{Var}_{\mathsf{AM}} = 2^k \cdot \binom{m}{k} \cdot \mathrm{Var}_0 \tag{6.2}$$

When k is relatively small (and hence $\binom{m}{k}$ is small), AM performs better than FC; when k is large, AM could perform worse than FC. Another limitation of this method is that one has to specify the value k ahead of time. After the protocol is executed, there is no way to answer any t-way marginal queries for $t > k$.

6.1.5 Fourier Transformation Method (FT)

Fourier Transformation (FT) was used for publishing k-way marginals in the centralized setting [27]. Kulkarni et al. [25] applied the technique to the local setting. Effectively, it is an optimization of the AM method. The motivation underlying FT is that, the calculation of a k-way marginal requires only a few coefficients in the Fourier domain. Thus, users can submit noisy Fourier coefficients that are needed to compute the desired k-way marginals, instead of values in those marginals.

This method results in slightly lower variance than AM. However, in order to reconstruct all k-way marginals, a large number of coefficients need to be estimated; thus this method would still perform poorly when k is large. Furthermore, the method is designed to deal

with the binary attributes. Therefore, the non-binary attributes must be pre-processed to binary attributes, resulting in more dimensions. For example, an attribute with d values has to be transformed into $\lceil \log_2 d \rceil$ binary attributes.

Here, we briefly analyze its variance. Specifically, there are $\sum_{s=0}^{k} \binom{d}{s}$ coefficients to be estimated. Estimating $\mathsf{T}_A(v)$ requires information for a selected set of 2^k coefficients, each multiplied by 2^{-k}. Therefore, this method has variance

$$\mathrm{Var}_{\mathsf{FT}} = \sum_{s=0}^{k} \binom{d}{s} \cdot \mathrm{Var}_0 \tag{6.3}$$

6.1.6 Expectation Maximization Method (EM)

This method allows each user to upload the value for each attribute separately with split privacy budget. The aggregator then conducts Expectation Maximization (EM) algorithm to reconstruct the marginal tables. This approach is first introduced by Fanti et al. [14] for estimating joint distribution for two attributes, and then generalized by Ren et al. [26] to handle multiple attributes.

Specifically, denote $y^j = \langle y_1^j, y_2^j, \ldots, y_m^j \rangle$ as the report from user j. The algorithm attempts to guess the private value distribution T_A, for any A, by maximizing the probability y^j are reported from user j.

The original EM algorithm runs slowly. Therefore, we use the algorithm proposed in the appendix of [12] to help compute T_A. In most cases, if the initial values are set using the result returned by this algorithm, the EM algorithm finishes quickly. Specifically, this algorithm first estimates the value distribution for any single attribute, and then uses that estimation to estimate distribution for any pair of attributes, and so on. The method is proven to produce unbiased estimation.

Overall, the EM method has the advantage of being able to compute t-way marginals for any t. But since ϵ is split into each attribute, this method has large variance.

6.2 CALM: Consistent Adaptive Local Marginal

In this section, we describe our proposed method CALM (Consistent Adaptive Local Marginal) for publishing k-way marginal via LDP. Our method is inspired by the PreView method for publishing marginal under the centralized DP setting [28], so we describe PreView first.

6.2.1 An Overview of PreView

The PreView method was designed for privately computing arbitrary k-way marginals for a dataset with m binary attributes in the centralized setting. PreView privately publishes a synopsis of the dataset. Using the synopsis, it can reconstruct any k-way marginal. The synopsis takes the form of t size-s marginals that are called *views*. Below we give an overview of the PreView method, using an example where there are $m = 8$ attributes $\{a_1, a_2, \cdots, a_8\}$, and we aim to answer all 3-way marginals. PreView has the following four steps. (See [28] for complete specification of PreView.)

Choose the Set of Views. The first step is to choose which marginals to include in the private synopsis as views. That is, one needs to choose t sets of attributes. PreView chooses these sets so that each size-2 (or size-3) marginal is covered by some view. For example, if aiming to cover all 2-way marginals, then one could choose the following $m = 6$ sets of attributes to construct views:

$$\{a_1, a_2, a_3, a_4\} \quad \{a_1, a_5, a_6, a_7\} \quad \{a_2, a_3, a_5, a_8\}$$
$$\{a_4, a_6, a_7, a_8\} \quad \{a_2, a_3, a_6, a_7\} \quad \{a_1, a_4, a_5, a_8\}$$

Observe that any pair of two attributes are included in at least one set.

Generate Noisy Views. In this step, for each of the t attribute sets, PreView constructs a noisy marginal over the attributes in the set, by adding Laplace noise $\mathcal{L}\left(\frac{m}{\epsilon}\right)$ to each cell

in the marginal table. This is the only step that needs direct access to the dataset. After this step, the dataset is no longer accessed.

Consistency Step. Given these noisy marginals/views, some 3-way marginals can be directly computed. For example, to obtain the 3-way marginal for $\{a_1, a_2, a_3\}$, we can start from the view for $\{a_1, a_2, a_3, a_4\}$ and marginalizes out a_4. However, many 3-way marginal are not covered by any of the 6 views. For example, if we want to compute the marginal for $\{a_1, a_3, a_5\}$, we have to rely on partial information provided by the 6 views. We can compute the marginals for $\{a_1, a_3\}$, $\{a_1, a_5\}$, and $\{a_3, a_5\}$, and then combine them to construct an estimation for $\{a_1, a_3, a_5\}$.

Observe that $\{a_1, a_5\}$ can be computed both by using the view for $\{a_1, a_5, a_6, a_7\}$ and by using the view for $\{a_1, a_4, a_5, a_8\}$. Since independent noises are added to the two marginals, the two different ways to compute marginal for $\{a_1, a_5\}$ most likely have different results. In addition, the noisy marginals may contain negative values. PreView performs constrained inference on the noisy marginals to ensure that the marginals in the synopsis are all non-negative and mutually consistent.

Generating k-way Marginals. From the t consistent views, one can reconstruct any k-way marginals. When given a set of k attributes, if all k attributes are included in one view, then we can compute the k-way marginal directly. When no view includes all k attributes, PreView uses Maximum Entropy estimation to compute the k-way marginal. For example, when given the marginals for $\{a_1, a_3\}$, $\{a_1, a_5\}$, and $\{a_3, a_5\}$, Maximum Entropy estimation finds among all possibles marginals for $\{a_1, a_3, a_5\}$ that are consistent with the three known marginals, the one with the maximum entropy. Note that while the marginal for $\{a_1, a_3, a_5\}$ have 7 unknowns (the 8 cells must sum up to 1), and each marginal over $\{a_1, a_3\}$, $\{a_1, a_5\}$, and $\{a_3, a_5\}$ gives 3 equations, these equations are not independent. In this case, the three 2-way marginals together give 6 independent linear constraints on the 7 unknowns, leaving one degree of freedom.

Discussions. Using the PreView method, one could answer k-way marginals for arbitrary k values. For a k-way marginal computed by PreView, there are two sources of errors. *Noise*

Errors are due to the Laplacian noises added to satisfy DP. *Reconstruction Errors* are due to the fact that one has to estimate a k-way marginal from partial information.

Two important algorithmic parameters affect the magnitude of these two kinds of errors. They are the number t of marginals/views in the synopsis, and the size s (i.e., number of attributes) of these views. With a larger s, the views cover more combinations of attributes, reducing Reconstruction Errors. However, one would be summing over more noisy entries to compute any marginal, increasing the Noise Errors. Similarly, a larger t means more marginals and better coverage of combinations of attributes, which reduces Reconstruction Errors. However, a larger t also means less privacy budget for each marginal and higher Noise Errors. Consider the running example with 8 attributes, by using 14 (instead of 6) size-4 marginals, one can ensure that any set of 3 attributes is covered by at least one of the marginals, eliminating Reconstruction Errors. However, this is done at the cost of adding noises sampled from $\mathcal{L}\left(\frac{14}{\epsilon}\right)$ instead of $\mathcal{L}\left(\frac{6}{\epsilon}\right)$ to each cell. Note that even if any set of 3 attributes is covered, answering 4-way marginals will still have Reconstruction Errors.

Analysis in [28] shows that the choice of optimal s (size of each marginal) is independent from parameters such as dataset size n, privacy parameter ϵ, and dimensionality m. In particular, setting s to be around 8 works well. The optimal choice of t (number of marginals), however, depends on n, ϵ, d, and the nature of the dataset. In [28], t is chosen to fully cover either all 2-way marginals or all 3-way marginals, using the concept of covering design.

6.2.2 Overview of the CALM Method

Ideas in the PreView method inspire the CALM method. In the LDP setting, we cannot compute a noisy marginal by adding Laplace noise to the true marginal, and we need to use FO protocols to do so. In PreView, data from all users are used for computing each of the t views, and the privacy budget is split into t equal portions. But in the local setting, several previous work pointed out that by partitioning users into groups, and having each group use the full privacy budget, the overall error will be smaller [5], [12], [29]. We adopt this design principle and split the user population into groups, with reports from users in each group to estimate one marginal.

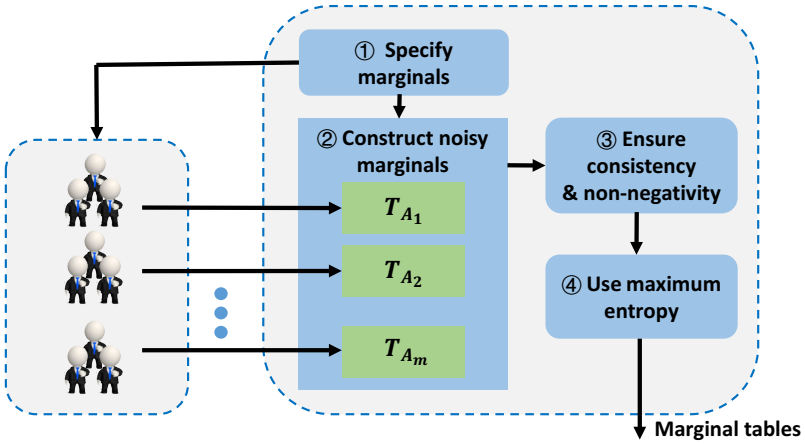


Figure 6.2. Illustration of CALM. The users to the left are partitioned into groups. The aggregator to the right first specifies the marginals to all the users and aggregate the reports for each marginal table. Then the aggregator process the data to publish the final results.

Figure 6.2 illustrates how CALM works. The aggregator first chooses a set of t marginals and the FO protocol to be used (e.g., GRR or OUE, note that we can also use OLH to save communication cost; but as the size of the marginals is typically not large, we use OUE to save server-side computation cost). The choice of whether to use GRR or OUE is determined by the number of cells in each marginal, because GRR is more accurate for domains of smaller size and OUE is better for larger domains. CALM adaptively chooses which of GRR and OUE to use, based on ϵ and the domain size for the marginals.

The aggregator then assigns each user to one of the marginals, and informs the user which marginal she should report. How this assignment is done is outside CALM. The aggregator can randomly partition the population into t groups of approximately the same size, and assigns users in one group to each marginal. Alternatively, the aggregator can use *public* information of the users (such as IP addresses) to help ensure that each group is representative of the overall population. It is also possible that the aggregator sends information of all t marginals to each user, having each user randomly select one and report on that marginal.

Each user projects her private value v onto the marginal she is reporting and reports the projected value of v via FO. On receiving users' reports, the server uses the aggregation

algorithm of FO to obtain the noisy marginal tables. Then the server processes the data via the consistency and reconstruction steps to obtain the final results, as in the PreView method.

One main challenge is how to choose the set of t marginals, and in particular the parameters s (size of each marginal) and t (number of marginals). The analysis for PreView in [28] is no longer valid for the local setting. We discuss this in Section 6.2.3. In addition, we want to deal with non-binary attributes, which we discuss in Section 6.2.4.

6.2.3 Choosing the Set of Marginals

The most important algorithmic parameters for CALM are the *marginal size* s, i.e., the number of attributes in each marginal, and the marginal number t, i.e., the number of different marginals. For ease of analysis, we assume all marginals are of equal size and receive equal number of users to contribute.

Similar to PreView, there are Noise Errors, which are caused by the addition of noises in the FO protocol, and Reconstruction Errors, which are caused by the fact that a k-way marginal may not be covered by any of the chosen marginal, and has to be estimated using the Maximum Entropy principle.

CALM has one additional source of errors that do not exist in PreView. CALM splits the user population into groups, and uses the marginal of one group as an estimation of the marginal of the whole population. Errors may be caused by the fact that the marginal of one randomly selected group is not representative of that for the whole population. We call these *Sampling Errors*. We analyze these errors below.

Noise Errors. To understand Noise Errors, we analyze the total variance of estimating 1-way marginals when they are included in at least one selected marginal, and how they are affected by the choice of t and s. For each s-way marginal table, there are $\frac{n}{m}$ users reporting it. By Equation (3.10), the variance for each cell is inversely proportional to the group size used to estimate it. More specifically, we have:

$$\mathrm{Var}_c = \left(\frac{4e^\epsilon}{(e^\epsilon-1)^2}, \frac{L-2+e^\epsilon}{(e^\epsilon-1)^2}\right) \cdot \frac{m}{n}$$

Here L is the number of cells in one marginal, and an s-way marginal with binary attributes has $L = 2^s$ cells. Note that when each attribute has different number of possible values, L is the expected number of cells in one marginal.

To construct a 1-way marginal from such an s-way marginal, each cell of the 1-way marginal is the summation of some cells from the larger (s-way) marginal. By linearity of variances, the variance for any 1-way marginal is $\text{Var}_1 = \text{Var}_c \cdot L$.

The above shows that increasing t adds a linear factor to the variance. However, increasing t also causes a 1-way marginal to be included more times. When a 1-way marginal is included t times, we can obtain t estimations of the 1-way marginal, one from each size-s marginal that includes it. Averaging these t estimations reduces the variance by a factor of t. More specifically, each size-s marginal includes s attributes. Therefore, in expectation, the information of each attribute will be contributed from $\frac{m \cdots}{d}$ s-way marginals. The average of these estimates are therefore

$$\begin{aligned}
&\mathsf{NE}(n, d, \epsilon, s) = \frac{\text{Var}_1}{\frac{m \cdot s}{d}} \\
=& \min\left(\frac{4e^\epsilon}{(e^\epsilon - 1)^2}, \frac{L - 2 + e^\epsilon}{(e^\epsilon - 1)^2}\right) \cdot \frac{m}{n} \cdot L \cdot \frac{d}{m \cdot s} \\
=& \min\left(\frac{4e^\epsilon}{(e^\epsilon - 1)^2}, \frac{L - 2 + e^\epsilon}{(e^\epsilon - 1)^2}\right) \cdot \frac{L}{s} \cdot \frac{d}{n}
\end{aligned} \tag{6.4}$$

The key observation here is that the magnitude of Noise Errors does not depend on t, which is different from PreView. It does depend on s and ϵ, where ϵ affects the first term, which is the variance of the FO protocol. The parameter s affects both the term $\frac{L}{s}$ and the variance for the FO protocol.

Also note that when we estimate k-way marginals based on the estimation of marginals of the k attributes, the estimation is affected by the errors for each of the k attributes, we thus use $k \cdot \mathsf{NE}(n, d, \epsilon, s)$ as the Noise Errors when we optimize for a particular k value.

Reconstruction Errors. Reconstruction Errors occur when a k-way marginal is not covered by any of the chosen marginal. The magnitude of Reconstruction Errors depends on to what extent attributes are correlated. If all attributes are mutually independent, then Reconstruction Errors do not exist. When attributes are dependent, the general trend is that

larger t and larger s will cover more combination of attributes, reducing reconstruction errors. The reduction effect of Reconstruction Errors diminishes as t increases. For example, if all k-ways marginals are already fully covered, Reconstruction Errors are already 0 and cannot be further decreased. Even if not all k-ways marginals are fully covered, increasing t beyond some reasonably large number will only cause diminishing return. Since Reconstruction Errors are dataset dependent, there is no formula for estimating them.

Sampling Errors. Sampling Errors occur when a marginal in a group of users deviates from the marginal in the whole population. The parameter s has no impact on Sampling Errors. However, increasing t would cause each group size $\frac{n}{m}$ to be smaller, raising Sampling Errors. When computing a marginal from a group of $s = n/m$ users, each cell in the marginal can be viewed as the sum of s independent Bernoulli random variables, divided by s. In other words, each cell is a binomial random variable divided by s. Thus each cell has variance $\frac{\mathsf{M}_A(v)(1-\mathsf{M}_A(v))}{s}$, where $\mathsf{M}_A(v)$ is the fraction of users with value v in the whole population. The Sampling Errors for an s-way marginal A are thus

$$\sum_{v \in V_A} \frac{\mathsf{M}_A(v)(1-\mathsf{M}_A(v))}{s} = \frac{m \times \sum_{c \in V_A} \mathsf{M}_A(v)(1-\mathsf{M}_A(v))}{n}$$

Since $\sum_{v \in V_A} \mathsf{M}_A(v) = 1$, we have $\sum_{v \in V_A} \mathsf{M}_A(v)(1-\mathsf{M}_A(v)) < \sum_{v \in V_A} \mathsf{M}_A(v) \cdot 1 = 1$. Thus the Sampling Errors are simply bounded by

$$\mathsf{SE}(n, m) = \frac{m}{n} \tag{6.5}$$

Choosing t and s. Both t and s affect Reconstruction Errors. In addition, t affects Sampling Errors, and s affects Noise Errors. Intuitively, we want to choose t and s to minimize the maximum of the three kinds of Errors, since the maximum would dominate the overall errors. However, we do not have a formula to estimate Reconstruction Errors, which is dataset dependent.

We propose to choose a *target error threshold* θ, which serves as a rough estimation of Reconstruction Errors when they are not zero, and choose t and s as follows:

- Compute the largest marginal size s_u, such that $k \cdot \mathsf{NE} < \theta$.

- When $s_u < k$, one chooses s_u and the largest t such that $\mathsf{SE} < \theta$.
- Otherwise, one chooses t and $s_t \in [k, s_u]$ such that the maximum of NE and SE is minimized.

While θ intends to be a rough estimation of Reconstruction Errors, it does not need to be chosen based on one particular dataset. One can run experiments with a public dataset of similar nature under different parameters, the best level of SSE that can be achieved is usually a good indicator of the magnitude of Reconstruction Errors. When a public dataset is unavailable, one can generate a synthetic dataset under some correlation assumption and run experiments. In experiments conducted for this paper, we choose $\theta = 0.001$, and use it for all datasets and settings.

Algorithm 2 Pseudocode to determine t and s

Input: Dataset parameters n, d, ϵ, k, error threshold θ.
Output: t and s.

procedure INFERENCE(n, d, ϵ, θ)
  Assign $m_u \leftarrow \theta \cdot n$, $s_u \leftarrow 2$
  while $k \cdot \mathsf{NE}(n, d, \epsilon, s_u + 1) \leq \theta$ **do**
    Increment $s_u \leftarrow s_u + 1$
  if $s_u < k$ **then**
    return $\min(m_u, \binom{d}{s_u}), s_u$
  Assign $s_b \leftarrow s_u$
  while $s_b > k$ and CoverDesign$(d, k, s_b - 1) \leq m_u$ **do**
    Decrement $s_b \leftarrow s_b - 1$
  if $s_b == s_u$ **then**
    return $\min(m_u, \binom{d}{s_u}), s_u$
  Assign $\mathsf{E} \leftarrow 1, m \leftarrow m_u, s \leftarrow s_u$
  for s_t in $[s_b, s_u]$ **do**
    Assign $m_t \leftarrow$ CoverDesign(d, k, s_t)
    if $\max(\mathsf{SE}(n, m_t), k \cdot \mathsf{NE}(n, d, \epsilon, s_t)) < \mathsf{E}$ **then**
      Update $\mathsf{E} \leftarrow \max(\mathsf{SE}(n, m_t), k \cdot \mathsf{NE}(n, d, \epsilon, s_t))$
      Update $m \leftarrow m_t, s \leftarrow s_t$
  return m, s

Algorithm 2 gives the pesudocode for determining t and s. The algorithm uses the formula to calculate Noise Errors NE from Equation (6.4), and Sampling Errors SE as in

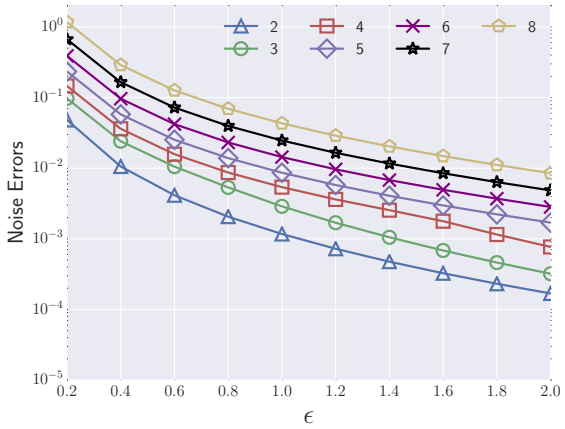


Figure 6.3. Noise Errors times k when $n = 2^{16}, m = 8, k = 3$.

Equation (6.5). CoverDesign is an external procedure to calculate the number of s-way marginals that can fully include all k-way marginals. Note that NE is for a single attribute; one can multiply NE by k to approximate the Noise Errors for the k-way marginals.

For example, Figure 6.3 gives the Noise Errors times k (i.e., $k \cdot \mathsf{NE}$) for $n = 2^{16}, m = 8$, and $k = 3$ when ϵ ranges from 0.2 to 2.0. If we fix $\theta = 10^{-3}$, we can read from the figure that when $\epsilon \leq 1.4$, only $s = 2$ can be used. Because larger s will make NE even larger; and we choose to allow some RE to exist. When ϵ is larger, e.g., $\epsilon = 2.0$, NE is already very small that we can tolerate more NE to eliminate RE. In this case, both $s = 3$ and $s = 4$ will give an $k \cdot \mathsf{NE} < \theta$, so the goal is to choose an s so that the maximum of $k \cdot \mathsf{NE}$ and SE is minimized. Specifically, when $s = 3$, CoverDesign gives $m = \binom{8}{3} = 56$ to cover all the 3-way marginals, rendering $\max(\mathsf{NE} = 0.00032, \mathsf{SE} = 0.00085) = 0.00085$; when $s = 4$, CoverDesign gives $m = 14$ (meaning that 14 4-way marginals suffice to cover all 3-way marginals when $m = 8$), thus giving $\max(\mathsf{NE} = 0.00076, \mathsf{SE} = 0.00021) = 0.00076$. Thus $s = 4$ and $m = 14$ is used.

6.2.4 Consistency between Noisy Marginals

When different marginals have some attributes in common, those attributes are actually estimated multiple times. Utility will increase if these estimates are utilized together. Specifically, assume a set of attributes A is shared by s marginals, $A_1, A_2, \ldots, A_s$. That is,

$A = A_1 \cap \ldots \cap A_s$. Now we can obtain s copies of T_A by summing from cells in each of the T_A's, i.e., $\mathsf{T}_{A_i}(v) = \sum_{v' \in V_{A_i}, v'_A = v_A} \mathsf{T}_{A_i}(v')$.

To obtain a better estimation of T_A, we use the weighted average of T_{A_i} for all marginal A_i. That is,

$$\mathsf{T}_A(v) = \sum_i w_i \cdot \mathsf{T}_{A_i}(v).$$

Since each T_{A_i} is unbiased, their average $\mathsf{T}_A(v)$ is also unbiased. To determine the distribution of the weights, the intuition is to put more weights to the more accurate estimations. Specifically, we minimize the variance of $\mathsf{T}_A(v)$, i.e., $\mathrm{Var}\left[\mathsf{T}_A(v)\right] = \sum_i w_i^2 \cdot \mathrm{Var}\left[\mathsf{T}_{A_i}(v)\right] = \sum_i w_i^2 \cdot L_i \cdot \mathrm{Var}_0$, where L_i is the number of cells from A_i that contribute to A, i.e., $L_i = |\{v' : v' \in V_{A_i}, v'_A = v_A\}|$, and Var_0 is the basic variance for estimating a single cell (we assume each marginal has a similar amount of users, but the analysis can be easily changed to different number of users). Formally, we have the following problem:

$$\begin{array}{ll} \mathsf{minimize} & \sum_i w_i^2 \cdot L_i \\ \mathsf{subject\ to} & \sum_i w_i = 1 \end{array}$$

According to KKT condition [30], [31], we can derive the solution: Define $L = \sum_i w_i^2 \cdot L_i + \mu \cdot (\sum_i w_i - 1)$, by taking the partial derivative of L for each of w_i, we have $w_i = -\frac{\mu}{2L_i}$. The value of μ can be solved by the equation $\sum_i w_i = 1$. As a result, $\mu = -\frac{2}{\sum_i \frac{1}{L_i}}$, and $w_i = \frac{\frac{1}{L_i}}{\sum_i \frac{1}{L_i}}$. Therefore, the optimal weighted average is

$$\mathsf{T}_A(v) = \frac{\sum_i \frac{1}{L_i} \cdot \mathsf{T}_{A_i}(v)}{\sum_i \frac{1}{L_i}}$$

Once the accurate T_A is obtained, all T_{A_i}'s can be updated. For any marginal A_i, we update all $v' \in V_{A_i}$ using the result of v where $v \in VA$ and $v'_A = v_A$. Specifically,

$$\mathsf{T}_{A_i}(v') \leftarrow \mathsf{T}_{A_i}(v') + \frac{1}{L_i}\left(\mathsf{T}_A(v) - \mathsf{T}_{A_i}(v)\right)$$

The remaining reconstruction operations are borrowed from PreView. After that, one can obtain the k-way marginals.

6.2.5 Discussion

We claim that CALM satisfies ϵ-LDP because all the information from each user to the server goes through an FO with ϵ as privacy budget, and no other information is leaked.

Although CALM is inspired from PreView, there are several differences between the two. Among the differences, many are because the two methods work under different privacy requirements. That is, PreView works in the centralized setting of differential privacy, while CALM works in the local setting. We summarize the differences as follows.

- In PreView, all the information are accessible to the server. The server operates on the dataset, adds noise, and then derive the answers. On the other hand, in CALM, each user sends noisy information to the server, who aggregates the reports, and then calculate the answers.
- CALM can handle non-binary datasets, while PreView is designed to handle only binary attributes.
- In PreView, each view is estimated through the information of all users, with split privacy budget. While in the local setting, it is known that it is better to partition users into groups. Therefore, in CALM, each marginal is estimated by only a group of users.
- Because of the above, CALM faces Sampling Errors, in addition to Noise Errors and Reconstruction Errors.
- In PreView, the number of marginals is critical and is dependent on the dataset. On the other hand, CALM is much less sensitive to the number of views (marginals).
- In PreView, the optimal view size does not depends on ϵ, and is around 8. However, in CALM, view size affects which FO protocol to be used and depends on ϵ.

6.3 Evaluation

We use experiments to empirically evaluate the effectiveness of our proposed method CALM, and to verify our analysis.

6.3.1 Experimental Setup

Our experimental setup is largely influenced by that in [25], which introduced the Fourier Transformation method and ran extensive comparisons of several methods for this problem.

Environment. All algorithms are implemented in Python 3.5 and all the experiments are conducted on a PC with Intel Core i7-4790 3.60GHz and 16GB memory.

Datasets. We run experiments on the following four datasets.

- POS [32]: A dataset containing merchant transactions of half a million users.
- Kosarak [10]: A dataset of click streams on a Hungarian website that contains around one million users.
- Adult [33]: A dataset from the UCI machine learning repository. After removing missing values, the dataset contains around 50 thousands records. The numerical attributes are bucketized into categorical attributes.
- US [34]: A dataset from the *Integrated Public Use Microdata Series* (IPUMS). It has around $40k$ records of the United States census in 2010.

The first two are transactional datasets where each record contains some items. We treat each item as a binary attribute. Thus these two datasets are binary. When running experiments with k binary attributes, we pre-process a dataset to include only the top m most frequent items. The later two are non-binary datasets, i.e., each attribute contains more than two categories.

Evaluation Methodology. To evaluate the performance of different methods, the *Sum of Squared Error* (SSE) of the marginals is reported. That is, we compute the ground truth and calculate the sum of squared difference in each cell. For each dataset and each method, we

choose 50 random k-way marginal queries and measure their SSE. This procedure is repeated 20 times, with result mean and standard deviation reported.

Competitors. The FC, AM, and EM methods can be directly applied. For a fair comparison, the FO used in those methods are also chosen adaptively.

The FT method is unable to deal with the non-binary attributes. Therefore, we implement the non-binary version of FT by encoding each non-binary attribute into several binary attributes.

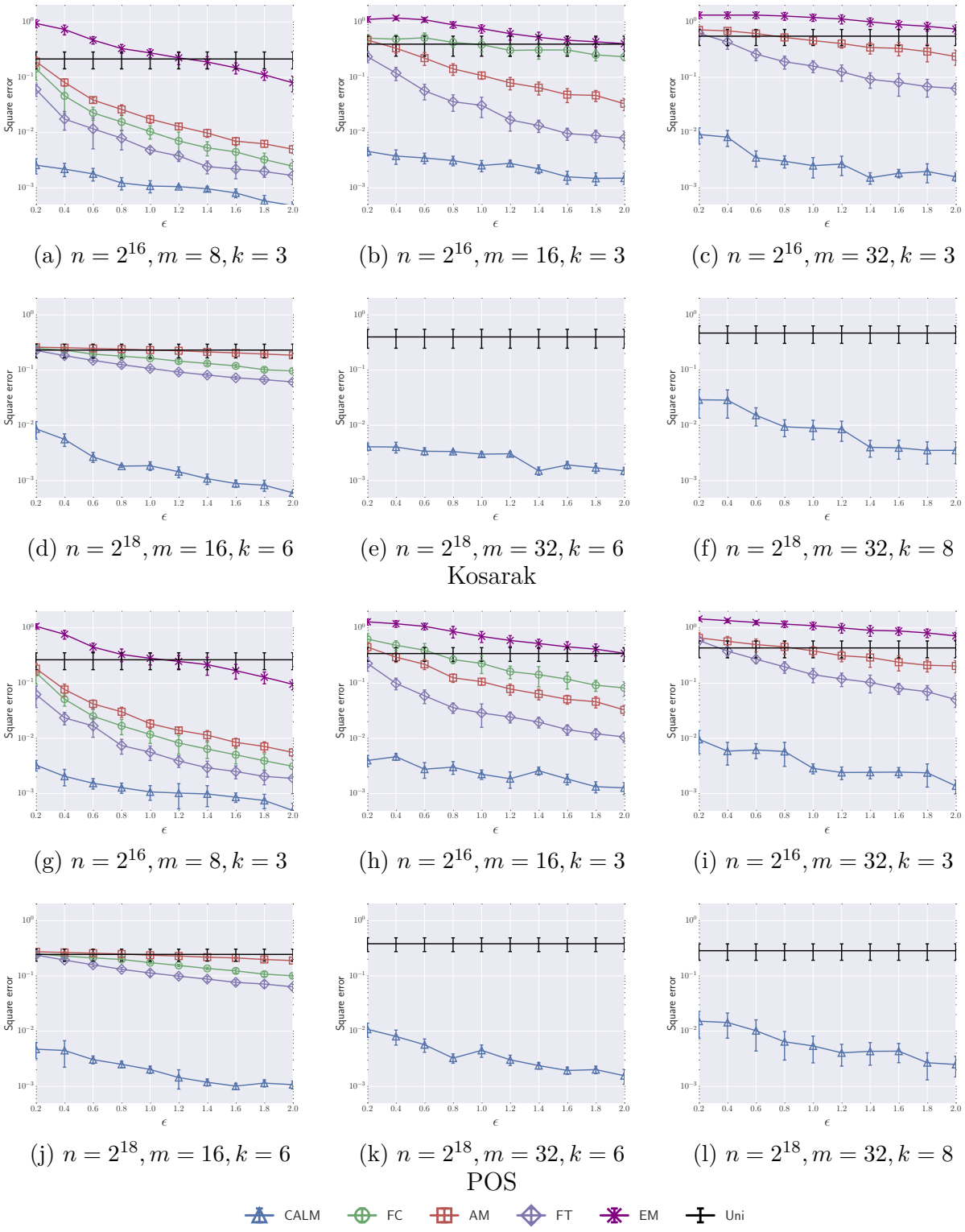
As a baseline comparison, we also plot the SSE of the Uniform method (Uni in the figures), which always returns a uniform distribution for any marginal tables. Clearly, if the performance of one method is worse than the Uniform method, the marginal constructed from that method is meaningless.

Experimental Settings. Different methods scale differently with respect to m, the number of attributes, and k, the size of marginals. Also, the error depends on n, the size of the dataset. We use three values of m: 8, 16, and 32. We consider $k = 3$ for all three settings of m. We consider $k = 6$ only for $d \in \{16, 32\}$, and $k = 8$ only for $m = 32$. This is because a larger k value makes more sense with a larger m value.

We consider two dataset sizes $n = 2^{16}$ and $n = 2^{18}$, which were used in [25]. Since all methods benefit similarly when n increases, the comparison results remain valid for other n sizes.

6.3.2 SSE on Binary Datasets

Figure 6.4 illustrates the results for comparing CALM against existing methods we discussed in Section 6.1 on two binary datasets Kosarak and POS.



(a) $n = 2^{16}, m = 8, k = 3$ (b) $n = 2^{16}, m = 16, k = 3$ (c) $n = 2^{16}, m = 32, k = 3$

(d) $n = 2^{18}, m = 16, k = 6$ (e) $n = 2^{18}, m = 32, k = 6$ (f) $n = 2^{18}, m = 32, k = 8$

(g) $n = 2^{16}, m = 8, k = 3$ (h) $n = 2^{16}, m = 16, k = 3$ (i) $n = 2^{16}, m = 32, k = 3$

(j) $n = 2^{18}, m = 16, k = 6$ (k) $n = 2^{18}, m = 32, k = 6$ (l) $n = 2^{18}, m = 32, k = 8$

Figure 6.4. Comparison of different methods on binary datasets. We only plot the methods that are scalable in each setting, Uni method is a baseline method. Results are shown in log scale.

In all settings, CALM significantly outperforms all existing algorithms, and the advantage of CALM increases for larger m and larger k values, and for smaller ϵ values. For most settings, the difference between CALM and FT, the closest competitor, is between one and two orders of magnitude. When ϵ is small, e.g., when $\epsilon = 0.2$, all existing algorithms perform close to the Uniform baseline, meaning they can provide very little information when the privacy budget is small. Whereas CALM can still provide enough information even for very small ϵ. Furthermore, many methods simply do not scale to the case of $m = 32$.

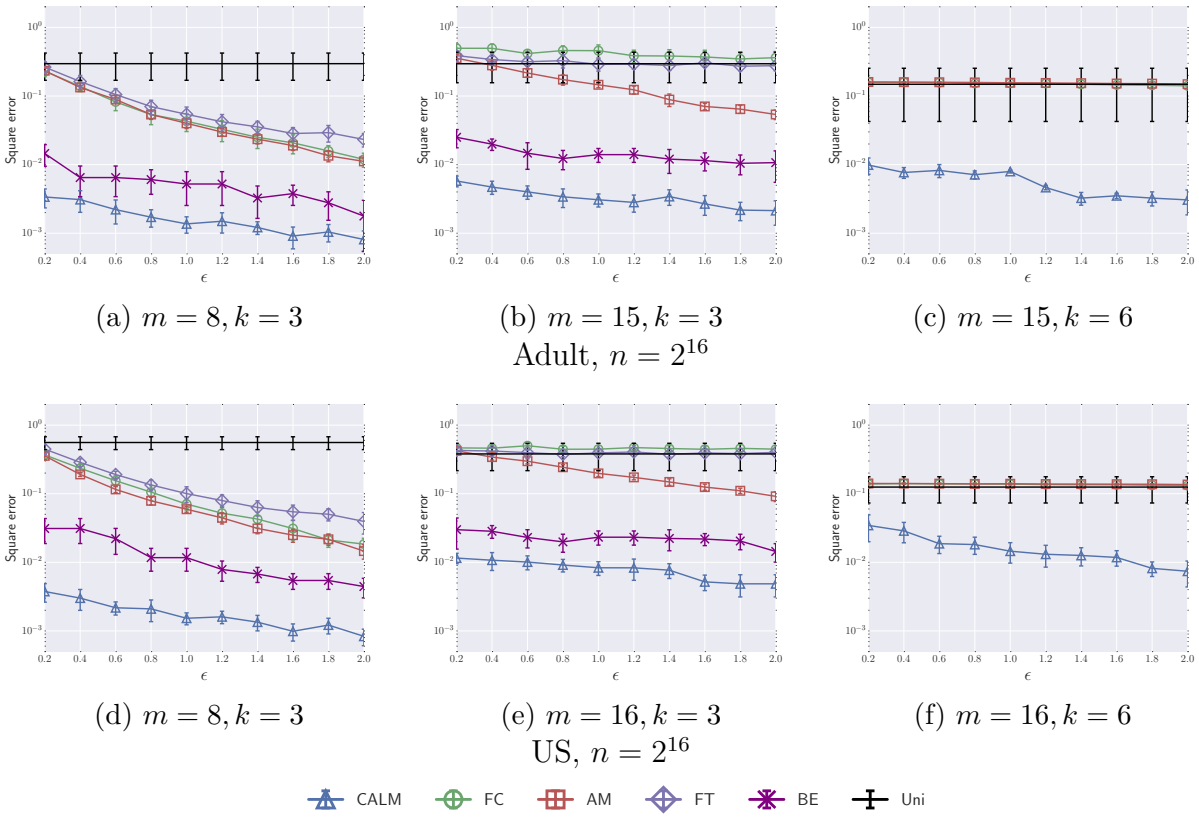
EM performs poorly, in fact it is often worse than the Uniform baseline. This is because EM requires each user to report information on all m attributes, in order to perform inference. This means dividing the privacy budget by m, which results in large perturbation. The other methods can split the population into groups, instead of splitting privacy budget, thus performing better. Also, when k is larger than 5, the computation time for EM method is too long to run efficiently (about 20 minutes each query). We thus do not plot EM for the $k = 6, 8$ cases.

Among the competitors, FT performs the best. When $m = 8, k = 3$, we can compute the variance for FC, AM and FT using Formulas (6.1), (6.2), and (6.3). The results are $256 \cdot \text{Var}_0$ for FC, $448 \cdot \text{Var}_0$ for AM, and $93 \cdot \text{Var}_0$ for FT. From Figures 6.4a and 6.4g, we can see that the experimental results match the analytical comparison.

For $m = 16$, CALM's performance is similar to the case of $m = 8$. Other methods, however, have significantly larger error. For example, in Figure 6.4b, when $\epsilon = 0.2$, the squared error of CALM is 0.0055, which is 41 times better than the state-of-the-art method, i.e., FT with squared error of 0.2266.

The performance of FC does not depends on k, since it constructs a full contingency table.

When $m = 32$, most of the existing methods are unable to scale, especially when $k = 8$. For the AM method, the number of possible marginals are $\binom{32}{8} = 10518300$. As a result, the average number of users that contribute information to each marginal is less than one when we choose $n = 2^{16}$ and 2^{18}. Similarly, the number of Fourier coefficients required to reconstruct 8-way marginals are $\sum_{s=1}^{8} \binom{32}{s} = 15033173$, resulting less than one user contributes to each coefficient.



(a) $m = 8, k = 3$ (b) $m = 15, k = 3$ (c) $m = 15, k = 6$

Adult, $n = 2^{16}$

(d) $m = 8, k = 3$ (e) $m = 16, k = 3$ (f) $m = 16, k = 6$

US, $n = 2^{16}$

Figure 6.5. Comparison of different methods in two non-binary datasets. We only plot the methods that are scalable in each setting, Uni method is a baseline method, BE method is the binary encoding version of CALM. Results are shown in log scale.

6.3.3 SSE on Non-binary Datasets

The experimental results for non-binary datasets, i.e., Adult and US, are shown in Figure 6.5. To reduce computational complexity, we pre-process all attributes to contain at most 3 categories.

The experimental results show the superiority of CALM, which achieves around 1 to 2 orders magnitude of improvement over existing methods.

By comparing the $m = 8$ and $k = 3$ setting in Figure 6.4 with Figure 6.5, we observe that FT performs better than FC and AM in the binary datasets, whereas performs worse in the non-binary datasets. The bad performance in the non-binary datasets is due to the binary

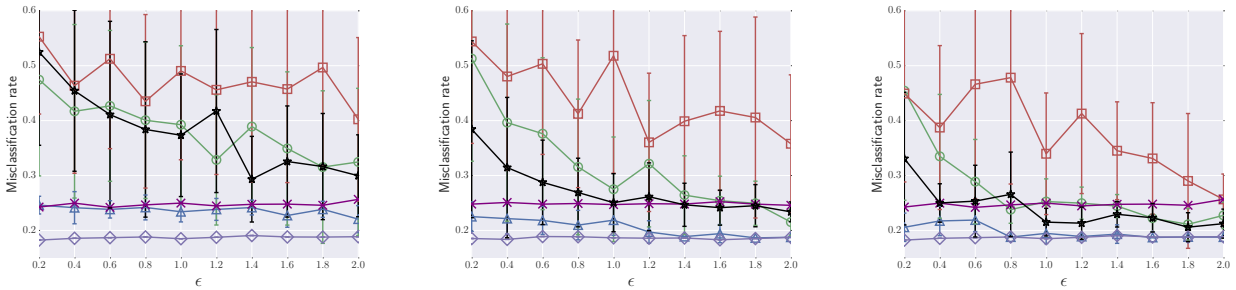
encoding process, which dramatically increases the number of Fourier coefficients required to reconstruct marginals. Considering the case where all the 8 attributes have 3 categories, m and k becomes 16 and 6 after the binary encoding. By variance analysis, the variance becomes $14893 \cdot \mathrm{Var}_0$, which is much larger than $93 \cdot \mathrm{Var}_0$ in the binary datasets.

To demonstrate the impact of handling non-binary attributes in CALM, we also utilize the idea of binary encoding to implement CALM, to which the consistent step of PreView can be directly applied. We call the binary encoding version of CALM the BE method. We observe that the CALM performs better than BE. The reason is that m and k becomes very large after binary encoding, which increase the variance. When $m = 16, k = 6$, the BE takes too much time in the Maximum Entropy estimation step. Thus, we do not plot BE in this case.

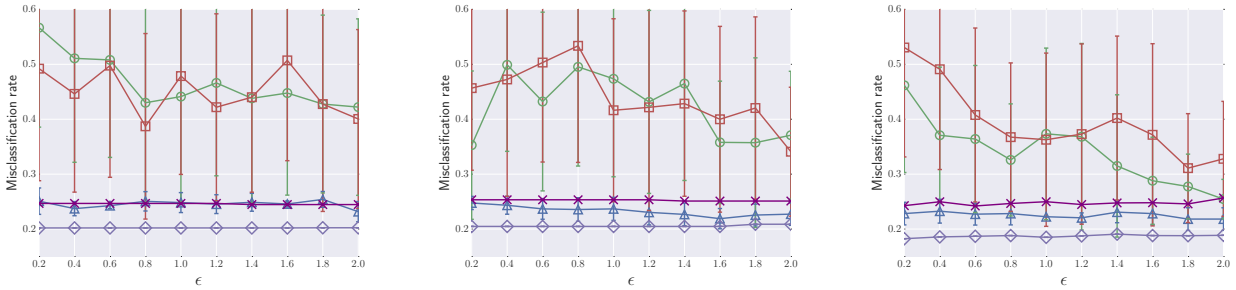
6.3.4 Classification Performance

To demonstrate the practical utility of the proposed method, we train the SVM classifiers using the Adult and US datasets. The goal of the classifier is to predict whether a user's annual income is above $50k$.

To train the model, we pick five attributes (features) and have each method output the 6-way marginals (five features plus the annual income label). The features for the Adult dataset are age, workclass, education, education-num, and occupation, as features; and the features for the US dataset are WRKRECAL (informed of work recall), GRADEATT (grade level attending), SCHLTYPE (public or private school), SCHOOL (school attendance), DIFFPHYS (ambulatory difficulty). Note that we pick features by their semantic relationships to the label. After the noisy marginal is obtained, a synthetic dataset is generated based on this marginal. The synthetic dataset is then used to train the SVM classifier. There are also two baselines: NoNoise represents the method without enforcing ϵ-LDP, it is the best case to aim for (using the same set of attributes). Majority represents the naive method that blindly predict the label by the majority label.

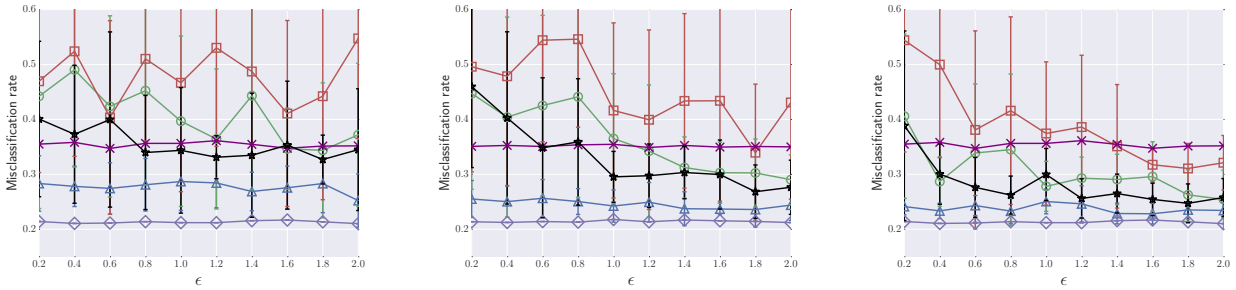


(a) Binary partition, $n = 2^{16}$ (b) Binary partition, $n = 2^{18}$ (c) Binary partition, $n = 2^{20}$

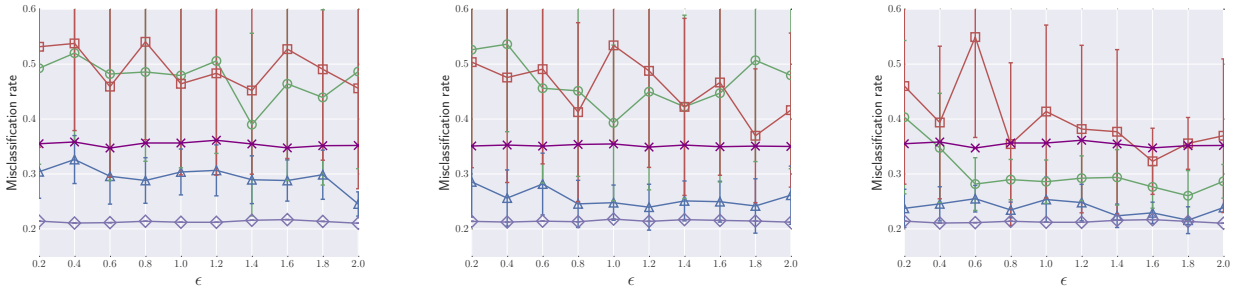


(d) Non-binary partition, $n = 2^{16}$ (e) Non-binary partition, $n = 2^{18}$ (f) Non-binary partition, $n = 2^{20}$

Adult, $m = 15, k = 6$



(g) Binary partition, $n = 2^{16}$ (h) Binary partition, $n = 2^{18}$ (i) Binary partition, $n = 2^{20}$



(j) Non-binary partition, $n = 2^{16}$ (k) Non-binary partition, $n = 2^{18}$ (l) Non-binary partition, $n = 2^{20}$

US, $m = 16, k = 6$

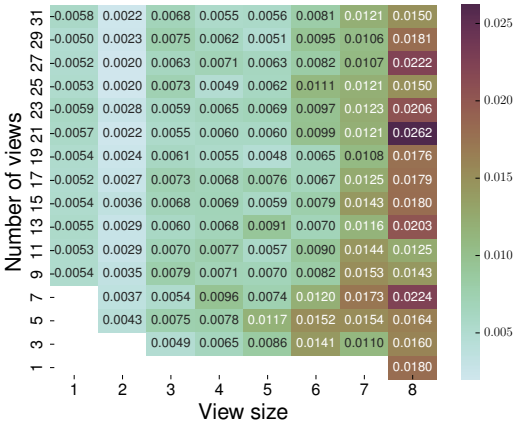
Figure 6.6. Comparison on classification performance. We only plot the methods that are scalable in each setting. NoNoise is the baseline where no noise is added; Majority is the naive method to always answer the majority label.

All the methods are evaluated following the typical process, where 80% of the records are sampled as training set, and the other 20% are used as the testing set. And we evaluate its utility by the *misclassification rate* on the testing set, i.e., the fraction of records in the testing set that are incorrectly classified. Figure 6.6 illustrates the misclassification rate of SVM classifier trained by different methods. It is shown that in most cases, the average misclassification rate of CALM is close to NoNoise. When ϵ is small, the classification model trained by FC and AM is not better than Majority (some times even worse than 50%, which is even worse than random guess, and thus useless).

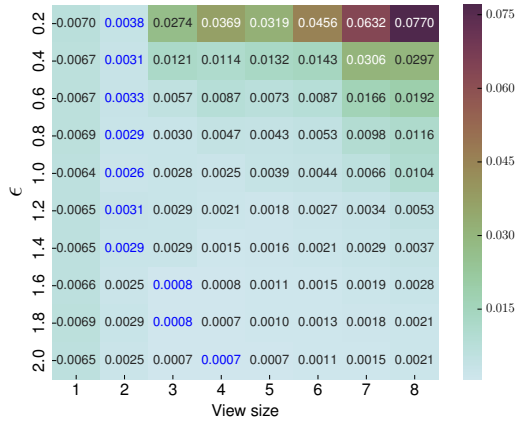
In the right two columns of Figure 6.6, we duplicate the datasets 4 times and 16 times to boost the accuracy. It can be seen that more users will help with accuracy. For example, for the binary case of the Adult dataset, the accuracy of CALM is almost optimal when $\epsilon = 1.4$ when the dataset is duplicated 4 times; while when the dataset is duplicated 16 times, the accuracy of CALM is almost optimal when $\epsilon = 0.8$. We also observe that when the dataset is non-binary (in the even rows of Figure 6.6), the performance is slightly worse than if the dataset is binary. This is because in the non-binary setting, there are more possible values in each attribute, thus making the result worse.

6.3.5 Verifying Marginal Parameters

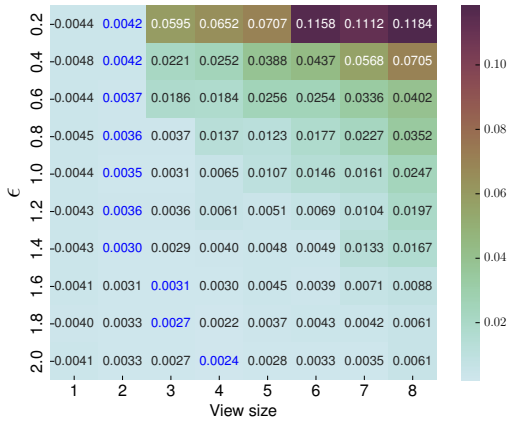
In Figure 6.7, we use heatmaps to illustrate the impact of marginal size s, number of marginals t and the privacy budget ϵ on the squared error.



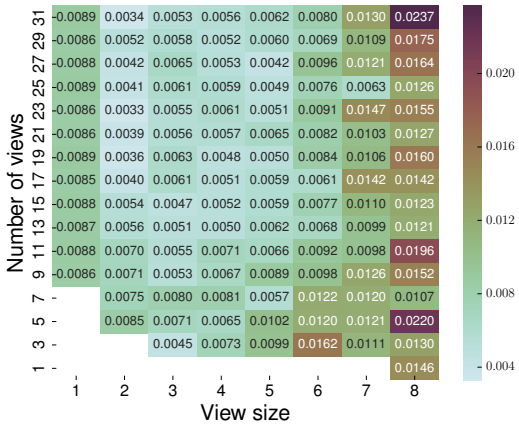
(a) POS, s vs t, $m = 8, k = 3, n = 2^{16}, \epsilon = 0.6$



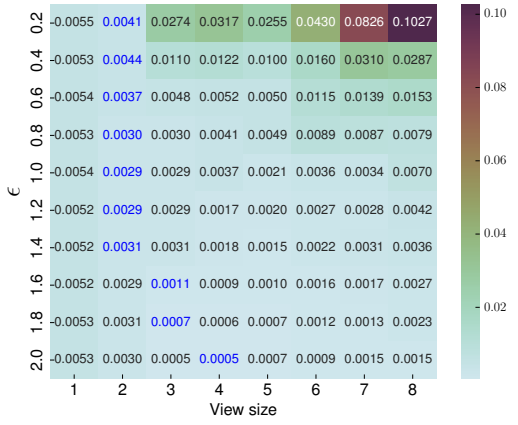
(b) POS, s vs ϵ, $n = 2^{16}, m = 8, k = 3, m = 16$



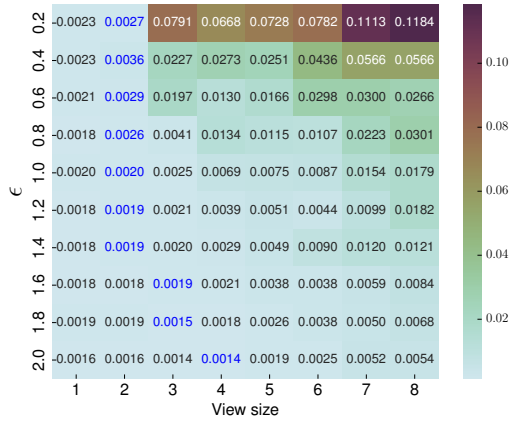
(c) POS, s vs ϵ, $n = 2^{16}, m = 16, k = 3, m = 16$



(d) Kosarak, s vs t, $m = 8, k = 3, n = 2^{16}, \epsilon = 0.6$



(e) Kosarak, s vs ϵ, $m = 8, k = 3, n = 2^{16}, m = 16$



(f) Kosarak, s vs ϵ, $m = 16, k = 3, n = 2^{16}, m = 16$

Figure 6.7. Mutual effects of marginal size s, number of marginals t and the privacy budget ϵ.

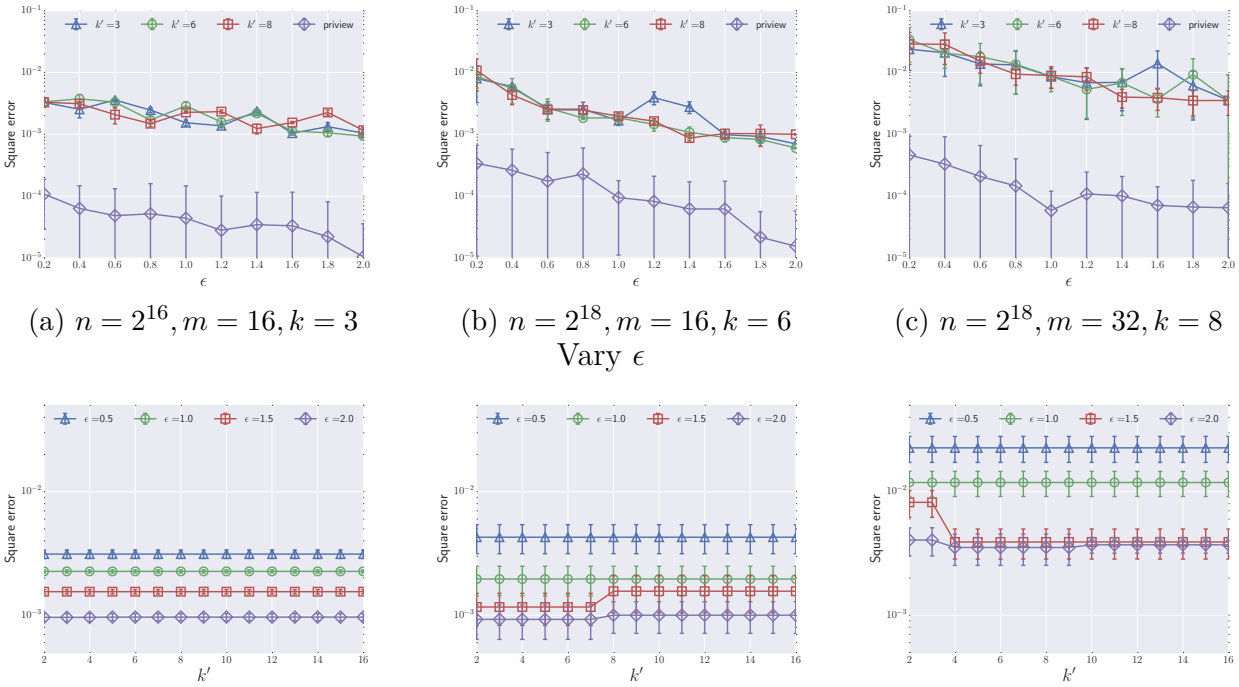
Figure 6.7a and 6.7d. shows the mutual effect of s and t on POS and Kosarak, respectively. This is for the setting of $m = 8, k = 3, \epsilon = 0.6$. The two heatmaps show that when s is fixed to a value other than 1 and 8, increasing t will gradually decrease the error, which is in accordance with the analysis in Section 6.2.3, as increasing t leads to covering more marginals, thus reducing Reconstruction Errors. While increasing t increases Sampling Errors, the level of Sample Errors even when $m = 32$ is around $2^{-11} = 0.0005$. Note that when $s = 1$, each marginal includes a single attribute, increasing t does not reduce Reconstruction Errors. Similarly, when $s = 8$ all 8 attributes are already covered in any marginal; increasing t thus does not change the error.

Figures 6.7b, 6.7c, 6.7e, and 6.7f. show the mutual effect of s and ϵ when t is fixed. We observe that when ϵ is small, it is better to choose smaller s. The reason is that in this case the noise dominates the error; thus we should choose smaller s to reduce noise. When ϵ is large, lager s is preferred since the effect of Reconstruction Errors is dominant. The blue numbers show the squared errors under the optimal setting through analysis, which is approximately approach to the experiment results.

6.3.6 Impact of k and the Local Setting

Figure 6.8 serves two purposes. One is to study the accuracy of CALM when one chooses parameters t and s that are optimized for k', but the query is for k-way marginals, where $k \neq k'$. This is interesting to know because one may want to support k-way marginals for different k values. The other is to compare the accuracy of CALM with the centralized setting of PreView, to understand how much utility one is giving up for the enhanced privacy of the local setting.

The first row of Figure 6.8 plots the effect of answering k-way marginals when optimized for $k' \in \{3, 6, 8\}$ and when using centralized PreView. When $k = 3$ (the left sub-figure), different settings of k' perform similarly. When $k = 6$ (the middle sub-figure), optimizing for $k' = 3$ clearly is worse when $\epsilon = 1.2$ and 1.4. This is because when optimizing for $k' = 3$, increasing ϵ from 1 to 1.2 causes s to increase from 2 to 3, because it is estimated that for



(a) $n = 2^{16}, m = 16, k = 3$ (b) $n = 2^{18}, m = 16, k = 6$ (c) $n = 2^{18}, m = 32, k = 8$

Vary ϵ

(d) $n = 2^{16}, m = 16, k = 3$ (e) $n = 2^{18}, m = 16, k = 6$ (f) $n = 2^{18}, m = 32, k = 8$

Vary k'

Figure 6.8. Kosarak dataset. Using t and s optimized for different k'.

3-way marginals, the noise error when going to $s = 3$ is sufficiently low at $\epsilon = 1.2$. However, when optimizing for $k' = 6$, the change of s from 2 to 3 occurs when $\epsilon = 1.6$. This also happens when $k = 8$ (the right sub-figure), where the setting of $k' = 3$ performs bad when $\epsilon = 1.6$ and 1.8. Overall, we generally see the best result when $k = k'$. Also, it appears that if one is unsure about k, the size of query marginals, one should optimize for a slightly larger k' value.

From first row of Figure 6.8, we also see that PreView performs one to two magnitudes better than CALM. This is mainly because much less noise is needed in the centralized setting. Theoretically, the amount of noise added in the local setting is $\Theta\left(\frac{1}{\sqrt{n}}\right)$, while in the centralized setting, the amount of noise is $\Theta\left(\frac{1}{n}\right)$.

The second row of Figure 6.8 plots the effect of answering k-way marginals while optimizing for a broader range of k' values (from 2 to 16). We consider $\epsilon \in \{0.5, 1.0, 1.5, 2.0\}$. When a setting results in choosing the same pair of m, s as another configuration, we reuse the

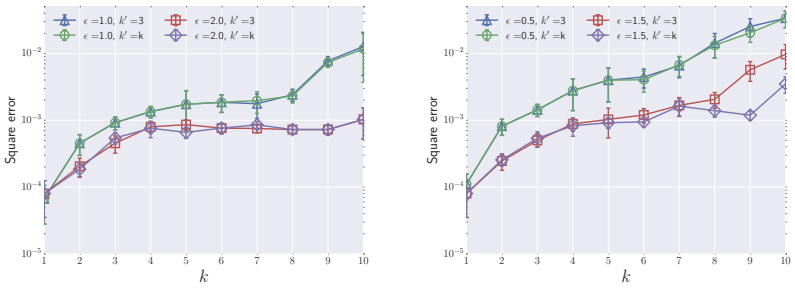


Figure 6.9. Kosarak dataset, $n = 2^{18}, m = 16$.

accuracy number in the plot instead of running the experiments again; thus any difference in a line is due to changes in m, s. We observe that when ϵ is small (i.e., $\epsilon \in \{0.5, 1.0\}$) the same parameters ($m = 65, s = 2$, to be precise) are chosen no matter which k' one is optimizing for. Also there is little difference in accuracy when computing $k = 3$. When $\epsilon = 1.5$ and $k = 6$, optimizing for $k' \geq 8$ is sub-optimal. In the right sub-figure, we observe that to compute $k = 8$-way marginals, optimizing for $k' \leq 3$ results in worse accuracy (especially for $\epsilon = 1.5$).

Figure 6.9 shows SSE for different k-way marginals fixing $n = 2^{18}$ and $m = 16$. For $\epsilon \in \{0.5, 1, 1.5, 2\}$, we plot results for two settings: t and s optimized for $k' = 3$; and t and s optimized for $k' = k$. We found that in most cases, the results for the two settings are similar, because the t and s settings are the same. For $\epsilon = 1.5$, when $k > 7$, the difference becomes significant. This is mainly because the s values are different: when $k' = 3$, $s = 3$ by Algorithm 2; but when $k' = k \in \{8, 9, 10\}$, $s = 2$ by Algorithm 2.

7. QUERY ANSWERING

(A version of this chapter has been published in ACM SIGMOD 2018 [35].)

We identify that local differential privacy (LDP) fits the class of analytical applications well. Suppose a number of individuals use a service in the cloud, and each user generates some *multi-dimensional data* during the service. Some dimensions, called *measure attributes*, about the service usage are naturally known to the service provider, e.g., active time and purchase amount (for the billing purpose); some other dimensions are *sensitive*, e.g., income and location, and users prefer to have them collected by the service provider in an privacy-preserving way; the remaining dimensions are non-sensitive. On the other side, the provider wants to analyze how the service performs by issuing analytical queries that aggregate measure attributes under constraints on sensitive dimensions. While all the dimensions will never be released to the public and the analytics are conducted internally by the provider, the provider needs to guarantee that the sensitive dimensions are handled properly by providing an LDP collection algorithm that runs on each user's device. A motivating example follows.

Example 1. *The multi-dimensional data model of users in an online shopping app is shown in Table 7.1. Users are anonymous. Measure attributes* ActiveTime *(how much time a user spent in the app) and* Purchase *(amount of money spent in the app), are inherently known to the service provider.* Age, Salary, *and* State *are sensitive dimensions, and attribute* OS *is non-sensitive. The service provider wants to analyze how much money is spent by a specific group of users using a query:*

$$\begin{aligned} &\mathsf{SELECT}\ \mathsf{SUM}(\mathsf{Purchase})\ \mathsf{FROM}\ T \\ &\mathsf{WHERE}\ \mathsf{Age} \in [30, 40]\ \mathsf{AND}\ \mathsf{Salary} \in [50\mathrm{K}, 150\mathrm{K}]. \end{aligned} \tag{7.1}$$

We study how to (approximately) answer a class of *multi-dimensional analytical (MDA) queries*, while each user's sensitive data is collected under LDP. An *MDA query* is a SQL query with aggregation (e.g., COUNT, SUM, or AVG) on measure attributes (accessible by service provider), and a predicate with equality and range constraints on sensitive dimensions (to be collected under LDP).

Table 7.1. A relational table T with sensitive dimensions

	Age	Salary	State	OS	ActiveTime	Purchase
	D_1	D_2	D_3	D_4	M_1	M_2
t_1	30	50K	NY	Win	1.6h	\$120
t_2	60	80K	WA	iOS	1.2h	\$100
t_3	50	90K	NY	Win	1.0h	\$100
t_4	40	70K	NY	iOS	1.8h	\$100

7.1 Preliminaries

7.1.1 Multi Dimensional Model and Analytics

Each *user* contributes a *tuple* t to a table with a set of attributes, called *dimensions* or *measures*. A dimension, denoted by D, appears in predicates, and a measure, denoted by M, is aggregated in analytical questions. We also use D or M to denote the *domain* (the set of possible values) of an attribute, and $t[D]$ and $t[M]$ are the attribute values in a tuple.

Multi-dimensional analytical (MDA) queries. Let T be the relational table, called *fact table*. We focus on the following class of *multi-dimensional analytical queries*:

$$Q_T(\mathsf{F}(M), \mathsf{C}): \qquad \mathsf{SELECT}\ \mathsf{F}(M)\ \mathsf{FROM}\ T\ \mathsf{WHERE}\ \mathsf{C} \tag{7.2}$$

- *Aggregation* F is COUNT$(*)$, SUM(M), or AVG(M), where M is a *public* measure. We focus on SUM(M) in the main text (COUNT is a special case and AVG can be derived from the other two). We will introduce how to use our solution for the other aggregate functions in Section 7.6.
- *Predicate* C consists of *point constraints* "$D_i = v_i$" for *categorical dimensions*, and *range constraints* "$D_i \in [l_i, r_i]$" for *ordinal dimensions*. We will first focus on conjunctions (AND-only) of one or more such constraints, and generalize our solutions for AND-OR expressions in Section 7.6.

7.1.2 Definition of LDP Revisited

A *server* collects tuples from users into T. In the introduced application scenarios, some dimensions are considered *sensitive* by users, and thus need to be collected in a privacy-preserving way; measures are public or known to the server (e.g., how much time a user spends on a service is known to the service provider for the billing purpose). More formally, suppose $D_1, \ldots, D_d$ are sensitive dimensions, we provide the following LDP guarantee:

Definition 7.1.1 (General ϵ-Local Differential Privacy)**.** *An algorithm $\mathcal{A}(\cdot)$ satisfies ϵ-local differential privacy (ϵ-LDP) if and only if for any, if for any pair of different tuples t and t, with $t[D_i] \neq t[D_i]$ for at least one $i \in \{1, \ldots, d\}$, we have,*

$$\forall T \subseteq Range(\mathcal{A}) : \ \mathsf{Pr}\left[\mathcal{A}(t) \in T\right] \leq e^{\epsilon} \cdot \mathsf{Pr}\left[\mathcal{A}(t) \in T\right].$$

Compared to the traditional Definition 2.2.1, this definition is more general but also realistic.

Example 2. *The multi-dimensional data model in Table 7.1 has six attributes. D_1 and D_2 are ordinal dimensions; D_3 and D_4 are categorical dimensions. There are also two numeric measures M_1 and M_2. The query in Example 1 is an MDA query with two range constraints on D_1 and D_2.*

D_1-D_3 are sensitive dimensions, and thus need to be collected under LDP. LDP guarantees that we cannot distinguish between two users with (30, 50K, NY) and (40, 70K, NY) based on their LDP reports, and thus, their dimension values are protected.

7.2 Weighted Frequency Oracle

We introduce building blocks used in our LDP mechanisms for MDA. We first introduce *weighted frequency queries* and their relationship to MDA. We then introduce how to estimate weighted frequencies if a random sample of users send their LDP reports – this twist will be used (in Sections 7.3-7.4) to boost the accuracy of MDA. Finally, a marginal-based solution for answering MDA is presented.

7.2.1 Weighed Frequency Queries and MDA

In a multi-dim data model, each user t has a private dimension $t[D] \in D$ and a public measure $t[M] \in \mathbb{R}$. A *weighted frequency query* asks, for a set of users S, what is the total measure of users with a given dimension value v, i.e.,

$$f_S^M(v; D) = \sum_{t \in S \wedge t[D]=v} t[M] \tag{7.3}$$

$$\Leftrightarrow \text{ SELECT } \mathsf{SUM}(M) \text{ FROM } S \text{ WHERE } D = v. \tag{7.4}$$

We write $f_S^M(v; D)$ as $f_S^M(v)$ if D is clear from the context.

Since $t[D]$ is private, each user uses an ϵ-LDP algorithm $\mathcal{A}_{\mathsf{FO}}$ to encode $t[D]$ as $\mathcal{A}_{\mathsf{FO}}(t[D])$ before sending it to the server. For a user set S, the server obtains an estimator $\hat{f}_S^M(v)$ of $f_S^M(v)$ from the LDP reports $\mathcal{A}_{\mathsf{FO}}(S) = \{\mathcal{A}_{\mathsf{FO}}(t[D])\}_{t \in S}$ and the public measure M. An LDP *weighted frequency oracle* refers to a pair of encoder and estimator $(\mathcal{A}_{\mathsf{FO}}, \hat{f}^M)$.

When $t[M] = 1$ for all users, $f_S^M(v)$ is equal to the (*unweighted*) frequency of v, $f_S(v)$, which is equivalent to a COUNT query:

$$f_S(v) = \sum_{t \in S \wedge t[D]=v} 1 = |\{t \in S \mid t[D] = v\}|$$

$$\Leftrightarrow \text{ SELECT } \mathsf{COUNT}(*) \text{ FROM } S \text{ WHERE } D = v. \tag{7.5}$$

Our Weighted Frequency Oracle $(\mathcal{A}_{\mathsf{FO}}, \hat{f}^M)$

We use an unweighted frequency oracle OLH (presented in Chapter 3) as a building block, and we show how to generalize it into a weighted frequency oracle.

The idea behind $(\mathcal{A}_{\mathsf{FO}}, \hat{f}^M)$ is to partition users into groups by their measures. Let $S_x = \{t \in S \mid t[M] = x\}$ be the group of users in S with measure equal to x. For a dimension value v, we establish the relationship between f and f^M via S_x:

$$f_S^M(v) = \sum_{\text{distinct } x} f_{S_x}^M(v) = \sum_{\text{distinct } x} x \cdot f_{S_x}(v). \tag{7.6}$$

We use an unweighted frequency oracle $(\mathcal{A}_{\mathsf{FO}}, \hat{f}_{S_x})$ to encode $t[D] = v$ in each S_x. We can then approximate $f_S^M(v)$ with the estimator $\hat{f}^M$ by combining the frequency estimates:

$$\hat{f}_S^M(v) = \sum_{\text{distinct } x} x \cdot \hat{f}_{S_x}(v), \tag{7.7}$$

Example 3. *Consider such a query against T in Table 7.1:*

SELECT SUM(Purchase) FROM T WHERE State = NY

The answer is $120+100+100+\ldots$. We have defined $S_{120} = \{t_1\}$ and $S_{100} = \{t_2, t_3, t_4\}$. The frequency of "NY" in S_{120}, $f_{S_{120}}(\text{NY}) = 1$, and, in S_{100}, $f_{S_{100}}(\text{NY}) = 2$. Thus, the answer can be also calculated $120 \times 1 + 100 \times 2 + \ldots$, and to estimate the answer, we can estimate $f_{S_{120}}(\text{NY})$ and $f_{S_{100}}(\text{NY})$ instead.

Proposition 7.2.1 (Weighted Frequency Oracle)**.** *Mechanism $(\mathcal{A}_{\mathsf{FO}}, \hat{f}^M)$ is ϵ-LDP. For a set of users S and a value $v \in D$, $\hat{f}_S^M(v)$ is an unbiased estimator of $f_S^M(v)$. Let $M_S^2 = \sum_{t \in S} t[M]^2$ and $M_S^2(v) = \sum_{t \in S \wedge t[D]=v} t[M]^2$. The error is*

$$\begin{aligned}\text{Err}(\hat{f}_S^M(v)) = \text{E}\left[\left(\hat{f}_S^M(v) - f_S^M(v)\right)^2\right] &= \frac{4M_S^2 e^\epsilon}{(e^\epsilon - 1)^2} + M_S^2(v) \\ &\leq \frac{M_S^2(e^\epsilon + 1)^2}{(e^\epsilon - 1)^2} = \text{O}\left(\frac{|S|\Delta^2}{\epsilon^2}\right) \text{ when } \epsilon \text{ is small,}\end{aligned}$$

where Δ is the range of M, i.e., $\Delta = \max(M) - \min(M)$. Moreover, estimation errors for two different values are additive:

$$\text{Var}\left[\hat{f}_S^M(u) + \hat{f}_S^M(v)\right] = \text{Var}\left[\hat{f}_S^M(u)\right] + \text{Var}\left[\hat{f}_S^M(v)\right].$$

Note that the above result does not depend on how large or how small each S_x is; in an extreme case, even if every distinct value of measure M appears only once ($|S_x| = 1$ for each x), we still have the same expected error.

7.2.2 Oracle Running on Random Samples

If we ask a random sample of users to report their private values $t[D]$ using $\mathcal{A}_{\mathsf{FO}}$, we can still estimate the weighted frequency of a value v in this sample using $\hat{f}^M$, and then scale the estimate up for the whole population – what is the accuracy loss in this procedure? This twist will be used in our mechanisms to boost its performance.

More formally, for a set of users S, we first randomly partition S into $S_1, \ldots, S_k$ (each user in S randomly chooses $i \in \{1, \ldots, k\}$, with equal probability $1/k$, and joins S_i). We run the weighted frequency oracle $(\mathcal{A}_{\mathsf{FO}}, \hat{f}^M)$ only on one sample, say, S_1. For a dimension value v, we can estimate its weighted frequency $f_S^M(v)$ in S using $\hat{f}^M$ on S_1. Define

$$\tilde{f}^M_{S,1/k}(v) = k \cdot \hat{f}^M_{S_1}(v), \tag{7.8}$$

where S_1 (or any of $S_1, \ldots, S_k$ generated above) is a random sample of S with sampling rate $1/k$. $\tilde{f}^M_{S,1/k}(v)$ in (7.8) is an unbiased estimator of weighted frequency $f_S^M(v)$, because

$$\mathrm{E}\left[\, \hat{f}^M_{S_1}(v) \,\right] = \mathrm{E}\left[\, \mathrm{E}\left[\, \hat{f}^M_{S_1}(v) \;\middle|\; S_1 \,\right] \right] = \mathrm{E}\left[\, [\,]\, S_1 \right] f_{S_1}(v) = \frac{1}{k} \cdot f_S(v).$$

The second equality is from the unbiasedness of $\hat{f}^M$ and the third one is due to the sampling process. The error in $\tilde{f}^M_{S,1/k}(v)$ comes from two sources, one due to LDP noise and the other due to sampling process. We can bound it as follows.

Proposition 7.2.2 (Accuracy Loss on Samples)**.** *$\tilde{f}^M_{S,1/k}(v)$ is an unbiased estimator of $f_S^M(v)$, and the error is bounded as*

$$\begin{aligned} &\mathrm{Err}(\tilde{f}^M_{S,1/k}(v)) = \frac{4kM_S^2 e^\epsilon}{(e^\epsilon - 1)^2} + (2k-1)M_S^2(v) \\ \leq &\frac{2kM_S^2(e^{2\epsilon}+1)}{(e^\epsilon - 1)^2} = \mathrm{O}\left(\frac{k|S|\Delta^2}{\epsilon^2}\right) \;\; \textit{when } \epsilon \textit{ is small,} \end{aligned}$$

where Δ is the range of M, i.e., $\Delta = \max(M) - \min(M)$.

7.2.3 Answering MDA via LDP Marginals

Mechanisms to estimate LDP marginals (Chapter 6) focus on COUNT queries. However, they can be adapted to handle MDA queries via a transition that is similar to (7.6).

$(\mathcal{A}_{\mathsf{MG}}, \mathsf{P}_{\mathsf{MG}})$: To answer an MDA query $Q_T(\mathsf{F}(M), \mathsf{C})$ in (7.2), we first partition T by measure M into sub-tables $T_x = \{t \in T \mid t[M] = x\}$. We use marginals estimated under LDP to count how many tuples in T_x satisfy the predicate C as $\bar{n}_x$ by summing up cells in the marginal on dimensions in C. For a SUM query Q_T, its answer can be estimated as $\sum_x x \cdot \bar{n}_x$.

Let's consider the SUM query in Example 1. We partition T by Purchase. In a sub-table, e.g., $T_{\$100}$, estimate the marginal

$$\text{SELECT COUNT}(*) \text{ FROM } T_{\$100} \text{ GROUP BY Age, Salary.}$$

Sum up (11×101) rows in the above 2-way marginal with $\mathsf{Age} \in [30, 40] \wedge \mathsf{Salary} \in [50\text{K}, 150\text{K}]$ to obtain $\bar{n}_{\$100}$, which contributes a term $(\$100 \cdot \bar{n}_{\$100})$ in the estimated answer.

Error analysis. We can analyze errors in the above marginal-based solution for data with one sensitive dimension using Proposition 7.2.1. Let D be a sensitive ordinal dimension, and we want to handle MDA queries with range constraints,

$$q : \text{SELECT SUM}(M) \text{ FROM } T \text{ WHERE } D \in [l, r].$$

We partition T by M. For each distinct $x \in M$, in the estimated LDP marginal of T_x on the dimension D, we sum up rows with $D \in [l, r]$, each contributing x in the answer. Since a 1-way marginal can be optimally estimated with a frequency oracle, the estimated answer to q is equivalent to:

$$\sum_{\text{distinct } x} \Big(\sum_{v \in [l,r]} x \cdot \hat{f}_{T_x}(v) \Big) = \sum_{v \in [l,r]} \hat{f}_T^M(v). \tag{7.9}$$

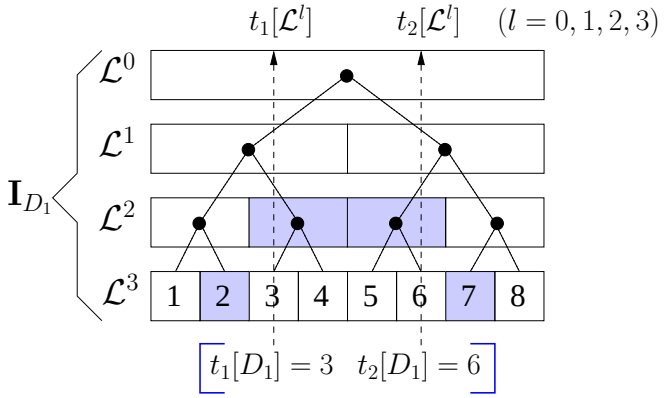


Figure 7.1. Hierarchy of intervals and the HI mechanism

The error now depends on how many distinct values of D we have within $[l, r]$. If D has m distinct values, from Proposition 7.2.1, the error of the above estimation is:

$$(r - l + 1) \cdot \mathrm{Err}(\hat{f}_T^M) = \Theta\Big(m|T|\Delta^2/\epsilon^2\Big) \tag{7.10}$$

with a linear dependency on $r - l + 1$ or m.

Suppose there are d sensitive dimensions, each with m distinct values, the worst-case error in the above solution is proportional to m^d, as we may need to sum up m^d marginal rows under range constraints on these dimensions.

In Sections 7.3-7.4, we propose new mechanisms to remove the linear/polynomial dependency on m in the error, via careful query decomposition and privacy-budget partitioning. Their error is poly-logarithmically dependent on m.

7.3 MDA with One Private Dimension

7.3.1 Hierarchical-Interval (HI) Mechanism

We propose a mechanism $(\mathcal{A}_{\mathsf{HI}}, \mathsf{P}_{\mathsf{HI}})$, whose one-dimensional version is inspired by the structure of a binary search tree, to ensure that the error is sublinear in m. Similar structures have also been used by previous work, e.g., [17], [36], to answer range counting queries in the centralized DP setting. We focus on the one-dimensional version in this section, and will generalize it to multi-dimensional MDA in Section 7.4.

Hierarchy of intervals. Suppose the ordinal dimension D has m distinct values, in the order of $z_1, z_2, \ldots, z_m$. We construct a hierarchical collections of intervals with a *fan-out* b, which can be viewed as a *perfect b-way tree*: each node corresponds to an interval, and has b children (except leaves), corresponding to b equally sized subintervals. We assume $m = b^h$ (if not, we can add some dummy values in D).

Level 0 in the hierarchy is $\mathcal{L}^0 = \{[z_1, z_m]\}$. $[z_1, z_m]$ corresponds to the root, and is recursively partitioned into b equally sized subintervals until we reach the leaves, i.e., intervals with unit length $\mathcal{L}^h = \{[z_1, z_1], \ldots, [z_m, z_m]\}$. There are b^l intervals on *level* l, each covering m/b^l values:

$$\mathcal{L}^l = \{[z_{(i-1)\cdot m/b^l+1}, z_{i\cdot m/b^l}] \mid i = 1, 2, \ldots, b^l\}.$$

Let $\mathbf{I}_D = \{\mathcal{L}^0, \ldots, \mathcal{L}^h\}$ be the whole hierarchy ($h = \log_b m$).

Example 4. *Consider an ordinal dimension D_1 with 8 values. Figure 7.1 shows its hierarchical intervals (with $b = 2$).*

Query rewriting with HI. In a query $q = Q_T(\mathsf{SUM}(M), D \in [l, r])$, the interval $[l, r]$ can be decomposed into $2(b-1)\log_b m$ (or less) disjoint intervals, $I^1, \ldots, I^p$, in the hierarchy $\mathbf{I}_D$. q can be decomposed into sub-queries on these intervals. If every user tells the server whether her/his dimension value is in each interval in $\mathbf{I}_D$, in an LDP way, each sub-query $Q_T(\mathsf{SUM}(M), D \in I^i)$ can be estimated ($i = 1, 2, \ldots, p$). The query q can be answered by assembling estimates for the $p \leq 2(b-1)\log_b m$ sub-queries, and thus with a polylogarithmic factor in the error. A rewriting example follows.

Example 5. *Assume that the ordinal dimension D_1 in Table 7.1 takes values in {1, 2, …, 8}. Consider the query*

$$q_1 : \mathsf{SELECT}\ \mathsf{SUM}(M_1)\ \mathsf{FROM}\ T\ \mathsf{WHERE}\ D_1 \in [2, 7].$$

$[2, 7]$ is decomposed into 4 intervals (the blue ones in Figure 7.1); correspondingly, q_1 is rewritten as the sum of four queries with "$D_1 \in [2, 2]$", "$D_1 \in [3, 4]$", "$D_1 \in [5, 6]$", and "$D_1 \in [7, 7]$".

In a naive implementation of the above strategy, each user sends $\Theta(m)$ LDP reports (as there are $\Theta(m)$ intervals in $\mathbf{I}_D$). We will show that, in fact, $\Theta(\log m)$ LDP reports suffice.

The main idea is that the dimension value $t[D]$ of a user t belongs to exactly $h+1$ intervals in $\mathbf{I}_D$, one on each level: suppose $t[D]$ belongs to $I^l \in \mathcal{L}^l$ on level l, we let $t[\mathcal{L}^l] = I^l$. We only need h frequency oracles, each encoding and collecting the interval $t[\mathcal{L}^l]$ on level l, for $l = 1, \ldots, h$.

In general, $[l, r]$ is partitioned into p disjoint intervals: $[l, r] = I^{k_1} \cup I^{k_2} \cup \ldots \cup I^{k_p}$ (I^{k_i} is on level $\mathcal{L}^{k_i}$). We rewrite

$$Q_T(\mathsf{SUM}(M), D \in [l, r]) = \sum_{i=1}^{p} Q_T(\mathsf{SUM}(M), D \in I^{k_i}) \tag{7.11}$$

where each sub-query $Q_T(\mathsf{SUM}(M), D \in I^{k_i})$ can be estimated by the weighted frequency oracle on $\mathcal{L}^{k_i}$ as $\hat{f}_T^M(I^{k_i})$.

Example 6. *In the hierarchy of D_1 in Figure 7.1, a tuple t_1 with $t_1[D_1] = 3$ belongs to one interval on each level (those crossed by the dashed arrowed line): $t_1[\mathcal{L}^3] = [3, 3]$, $t_1[\mathcal{L}^2] = [3, 4]$, $t_1[\mathcal{L}^1] = [1, 4]$, and $t_1[\mathcal{L}^0] = [1, 8]$. The first three intervals are encoded and collected using frequency oracles.*

As in Example 5, q_1 is decomposed into four sub-queries: the one with "$D_1 \in [3, 4]$" can be estimated with $\hat{f}_T^M([3, 4])$.

HI mechanism $(\mathcal{A}_{\mathsf{HI}}, \mathsf{P}_{\mathsf{HI}})$**.** On the client side, the privacy budget is partitioned evenly for the h levels $\mathcal{L}^1, \ldots, \mathcal{L}^h$, and the interval a tuple t belongs to on each level (i.e., $t[\mathcal{L}^i]$) is encoded using $\mathcal{A}_{\mathsf{FO}}$ in a weighted frequency oracle.

On the server, for a query q, we estimate each sub-query $Q_T(\mathsf{SUM}(M), D \in I^{k_i})$ in (7.11) using the weighted frequency estimator $\hat{f}^M(I^{k_i})$, and sum them up as an estimation to q.

$$\mathsf{P}_{\mathsf{HI}}(q) = \sum_{i=1}^{p} \hat{f}_T^M(I^{k_i}). \tag{7.12}$$

Theorem 7.3.1 (1D-HI). *i)* $\mathcal{A}_{\mathsf{HI}}$ *satisfies* ϵ*-LDP. ii)* $\mathsf{P}_{\mathsf{HI}}(q)$ *is an unbiased estimator of* q*, and the expected squared error*

$$\begin{aligned}\mathrm{Err}(\mathsf{P}_{\mathsf{HI}}(q)) &\leq 2(b-1)\log_b m \cdot M_T^2 \cdot \frac{(e^{\epsilon/\log_b m}+1)^2}{(e^{\epsilon/\log_b m}-1)^2} \\ &= \mathrm{O}\left(\frac{n\Delta^2 \log^3 m}{\epsilon^2}\right) \text{ when } \epsilon \text{ is small,}\end{aligned} \tag{7.13}$$

where $n = |T|$ *is the number of users,* Δ *is the range of* M*,* $M_T^2 = \sum_{t\in T} t[M]^2$*, and the constant* b *is the fan-out.*

7.3.2 Better Accuracy via Level Partitioning

In $\mathcal{A}_{\mathsf{HI}}$, the privacy budget ϵ is partitioned evenly for the h levels. An alternative is to randomly partition the users into h groups instead of partitioning the privacy budget: users in a group S_l, corresponding to level l, can be regarded as a random sample with sampling rate $1/h$, and spend all the privacy budget on only level l. A bit surprisingly, this alternative has the accuracy boosted by orders of magnitude. The intuition behind this is: we gain accuracy by spending more privacy budget on each level, but lose accuracy as each level is supported for a random sample of users (refer to Proposition 7.2.2 in Section 7.2.2); as long as the accuracy gain overcomes the loss, the overall accuracy can be boosted.

Query $q = Q_T(\mathsf{SUM}(M), D \in [l, r])$ is decomposed in the same way as (7.11). For a sub-query $Q_T(\mathsf{SUM}(M), D \in I^{k_i})$ in (7.11), we refer to the user group S_{k_i} corresponding to level k_i. As introduced in Section 7.2.2, we can run frequency oracles on the random sample S_{k_i}, and scale up the estimation $\hat{f}^M_{S_{k_i}}(I^{k_i})$ by a factor of h, to approximate the sub-query's answer. The mechanism $(\mathcal{A}_{\mathsf{HIO}}, \mathsf{P}_{\mathsf{HIO}})$ based on the above idea is described in Algorithm 3. Its error bound is given in Theorem 7.3.2.

Algorithm 3 1D HI Optimized $(\mathcal{A}_{\mathsf{HIO}}, \mathsf{P}_{\mathsf{HIO}})$

Client side: Encode private dimension $t[D]$.

1: Randomly pick $l \in \{1, 2, \ldots, h\}$ with equal prob.
2: Suppose $t[D]$ is in interval $I^l \in \mathcal{L}^l$: $t[\mathcal{L}^l] \leftarrow I^l$;
3: Create LDP report:

$$\mathcal{A}_{\mathsf{HIO}}(t) \leftarrow \left(l, \mathcal{A}^{\epsilon}_{\mathsf{FO}}(t[\mathcal{L}^l])\right). \tag{7.15}$$

Server side: MDA query $q = Q_T(\mathsf{SUM}(M), D \in [l, r])$.

1: Let user groups $S_l \leftarrow \emptyset$ for $l = 1, \ldots, h$.
2: For each user t. if we get $\left(l_0, \mathcal{A}^{\epsilon}_{\mathsf{FO}}(t[\mathcal{L}^{l_0}])\right)$:
3: $\quad S_{l_0} \leftarrow S_{l_0} + \{t\}$;
4: Decompose $[l, r]$ into p disjoint intervals $I^{k_1} \in \mathcal{L}^{k_1}$, $I^{k_2} \in \mathcal{L}^{k_2}$, ..., $I^{k_p} \in \mathcal{L}^{k_p}$ in the hierarchy $\mathbf{I}_D$.
5: For each $i = 1, 2, \ldots, p$:
6: $\quad$ Estimate $f_T^M(I^{k_i})$ as $\tilde{f}^M_{T,1/h}(I^{k_i})$ (using (7.8));

$$\tilde{f}^M_{T,1/h}(I^{k_i}) = h \cdot \hat{f}^M_{S_{k_i}}(I^{k_i}), \tag{7.16}$$

7: Output an estimation to q as:

$$\mathsf{P}_{\mathsf{HIO}}(q) = \sum_{i=1}^{p} \tilde{f}^M_{T,1/h}(I^{k_i}). \tag{7.17}$$

Theorem 7.3.2 (1D-HIO)**.** *i)* $\mathcal{A}_{\mathsf{HIO}}$ *satisfies* ϵ*-LDP. ii)* $\mathsf{P}_{\mathsf{HIO}}(q)$ *is an unbiased estimator of* q *with expected squared error*

$$\begin{aligned} \mathrm{Err}(\mathsf{P}_{\mathsf{HI}}(q)) &\le 4(b-1)\log_b^2 m \cdot M_T^2 \cdot \frac{(e^{2\epsilon}+1)}{(e^{\epsilon}-1)^2} \\ &= \mathrm{O}\left(\frac{n\Delta^2 \log^2 m}{\epsilon^2}\right) \ \textit{when } \epsilon \textit{ is small,} \end{aligned} \tag{7.14}$$

where $n = |T|$, Δ *is the range of* M*, and* $M_T^2 = \sum_{t \in T} t[M]^2$.

The HIO mechanism boosts the accuracy by a factor of $\log_b m$ in comparison to the HI mechanism (7.13). We use $b = 5$ in our implementation to minimize RHS of (7.14).

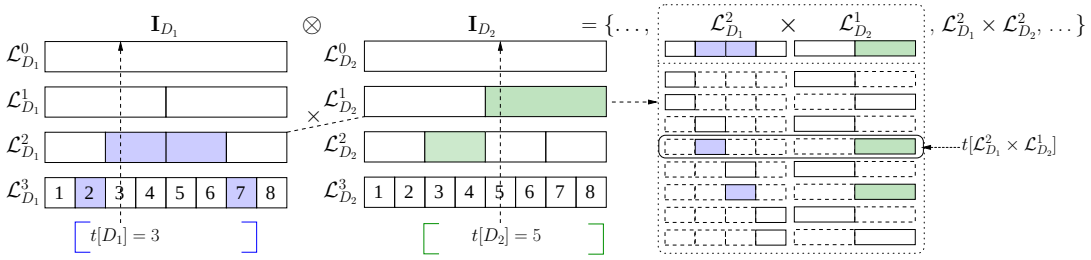


Figure 7.2. 2D hierarchy of intervals, query decomposition, and HI mechanism

7.4 Multiple Private Dimensions

We now introduce how to handle multiple private dimensions in MDA queries. We will first focus on the case when we have multiple ordinal dimensions and range constraints in an MDA query, for which we extend our HI/HIO mechanism in Section 7.3 to a multi-dimensional one. We will then introduce how to handle a combination of ordinal and categorical dimensions. Finally, when there are many sensitive dimensions in the data model and the worst-case error blows up, we will introduce a *split-and-conjunction* mechanism which is still able to handle low-dimensional queries.

7.4.1 Multiple Ordinal Dimensions

In order to extend our HI mechanism for multiple dimensions, let's first introduce a multi-dimensional hierarchy of intervals, which naturally generalizes the one-dimensional hierarchy in Section 7.3.1. An MDA query can be decomposed into a polylogarithmic (in dimension cardinalities) number of sub-queries in this hierarchy, and they together are aggregated to answer the original MDA query without blowing up the worst-case error. Similar user-partitioning techniques as in Section 7.3.2 can be applied to boost the accuracy.

Multi-dimensional Hierarchical Intervals

Recall that $\mathbf{I}_D = \{\mathcal{L}^0_D, \ldots, \mathcal{L}^h_D\}$ is the hierarchy for dimension D: level 0 is $\mathcal{L}^0_D = \{[z_1, z_m]\}$ and $\mathcal{L}^{l+1}_D$ is obtained by partitioning each interval in $\mathcal{L}^l_D$ into b equally sized subintervals.

W.l.o.g., assume each dimension has the same cardinality $m = b^h$ (i.e., # distinct values) for the simplicity of explanation.

Two-dimensional hierarchy. Let's first focus on two dimensions. Define a 2-*dim hierarchy* to be:

$$\mathbf{I}_{D_1} \otimes \mathbf{I}_{D_2} = \left\{ \mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2} \;\middle|\; 0 \le l_1, l_2 \le h \right\}.$$

Each $\mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2}$ is called a 2-*dim level*. There are a total of $(h+1)^2$ 2-dim levels in a 2-dim hierarchy. Each pair $\langle I_1, I_2 \rangle \in \mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2}$ is called a 2-*dim interval*. We will write $I_1 I_2 \dots I_d$ as a shorthand for $\langle I_1, I_2, \dots, I_d \rangle$ in the rest part.

Consider a tuple t, for each l_1 and l_2, $t[D_1]$ and $t[D_2]$ belong to exactly one interval in $\mathcal{L}_{D_1}^{l_1}$ and $\mathcal{L}_{D_2}^{l_2}$: let them be $I_1^{l_1}$ and $I_2^{l_2}$, respectively. We augment t with *a new dimension* "$\mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2}$" for the corresponding 2-dim level: let

$$t[\mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2}] = I_1^{l_1} I_2^{l_2},$$

which means that $t[D_1] \in I_1^{l_1} \wedge t[D_2] \in I_2^{l_2}$. Indeed, we have

$$\begin{aligned} & Q_T(\mathsf{SUM}(M), D_1 \in I_1^{l_1} \wedge D_2 \in I_2^{l_2}) \\ =& Q_T(\mathsf{SUM}(M), \mathcal{L}_{D_1}^{l_1} \times \mathcal{L}_{D_2}^{l_2} = I_1^{l_1} I_2^{l_2}) = f_T^M(I_1^{l_1} I_2^{l_2}). \end{aligned}$$

In an MDA query $q = Q_T(\mathsf{SUM}(M), D_1 \in [l_1, r_1] \wedge D_2 \in [l_2, r_2])$, $[l_1, r_1]$ can be decomposed into p_1 disjoint intervals in $\mathbf{I}_{D_1}$: $[l_1, r_1] = I_1^1 \cup \ldots \cup I_1^{p_1}$, and similarly $[l_2, r_2] = I_2^1 \cup \ldots \cup I_2^{p_2}$. We can then decompose q into $p_1 \times p_2$ sub-queries:

$$\begin{aligned} & Q_T(\mathsf{SUM}(M), D_1 \in [l_1, r_1] \wedge D_2 \in [l_2, r_2]) \\ =& \sum_{1 \le a \le p_1, 1 \le b \le p_2} Q_T(\mathsf{SUM}(M), D_1 \in I_1^a \wedge D_2 \in I_2^b) \\ =& \sum_{1 \le a \le p_1, 1 \le b \le p_2} f_T^M(I_1^a I_2^b). \end{aligned} \tag{7.18}$$

Since $p_1, p_2 \leq 2(b-1)\log_b m$, there are $\mathrm{O}(\log^2 m)$ sub-queries. Each sub-query can be estimated as $\hat{f}_T^M(I_1^a I_2^b)$ using a weighted frequency oracle on the corresponding 2-dim level, and then we just need to sum up these estimates to answer q.

Example 7. *Suppose the two ordinal dimensions D_1 and D_2 take values in $\{1, 2, \ldots, 8\}$. A 2-dim hierarchy on them is shown in Figure 7.2. Their individual 1-dim hierarchies $\mathbf{I}_{D_1}$ and $\mathbf{I}_{D_2}$ are on the left, each with 4 levels. The 2-dim hierarchy is a Cartesian product of the two, with 4×4 2-dim levels. In particular, the 2-dim level, $\mathcal{L}^2_{D_1} \times \mathcal{L}^1_{D_2}$, depicted on the right, is a Cartesian product of two 1-dim interval sets $\mathcal{L}^2_{D_1}$ and $\mathcal{L}^1_{D_2}$, with 4×2 2-dim intervals, each of which is a pair of 1-dim intervals, with one from $\mathcal{L}^2_{D_1}$ and the other from $\mathcal{L}^1_{D_2}$. For example, the 4th one (top-to-bottom) in the figure is $[3, 4][5, 8]$.*

Consider a tuple t with $t[D_1] = 3$ and $t[D_2] = 5$. It belongs to the above 2-dim interval as $t[D_1] \in [3, 4]$ and $t[D_2] \in [5, 8]$. Thus, the augmented dimension $t[\mathcal{L}^2_{D_1} \times \mathcal{L}^1_{D_2}] = [3, 4][5, 8]$.

Consider the following MDA query:

$$q_2 \text{ :SELECT SUM}(M_1) \text{ FROM } T$$
$$\text{WHERE } D_1 \in [2, 7] \text{ AND } D_2 \in [3, 8]$$

As shown in Figure 7.2, $[2, 7]$ can be partitioned into 4 intervals in $\mathbf{I}_{D_1}$ (blue ones), and $[3, 8]$ partitioned into 2 in $\mathbf{I}_{D_2}$ (green ones). Thus, q_2 can be decomposed into 4×2 disjoint sub-queries: e.g., two on the 2-dim level $\mathcal{L}^2_{D_1} \times \mathcal{L}^1_{D_2}$, one with "$D_1 \in [3, 4]$ AND $D_2 \in [5, 8]$" and one with "$D_1 \in [5, 6]$ AND $D_2 \in [5, 8]$". If each user reports all the augmented dimensions with LDP weighted frequency oracles, answers to such sub-queries (with 2-dim intervals in their predicates) can be approximated.

***d*-dim hierarchy.** The construction of a 2-dim hierarchy can be easily extended for more dimensions. Define:

$$\mathbf{I}_{D_1} \otimes \ldots \otimes \mathbf{I}_{D_d} = \left\{ \mathcal{L}^{l_1}_{D_1} \times \ldots \times \mathcal{L}^{l_d}_{D_d} \;\middle|\; 0 \leq l_1, l_2, \ldots, l_d \leq h \right\}$$

to be a *hierarchy* of *d-dim levels* (there are $(h+1)^d$ levels). Each $I_1 I_2 \dots I_d \in \mathcal{L}_{D_1}^{l_1} \times \dots \times \mathcal{L}_{D_d}^{l_d}$ is a *d-dim interval*.

An MDA query $q = Q_T(\mathsf{SUM}(M), D_1 \in [l_1, r_1] \wedge \dots \wedge D_d \in [l_d, r_d])$ can be thus decomposed into $p_1 \times \dots \times p_d$ sub-queries:

$$\begin{aligned} & Q_T(\mathsf{SUM}(M), D_1 \in [l_1, r_1] \wedge \dots \wedge D_d \in [l_d, r_d]) \\ = & \sum_{1 \le i_1 \le p_1, \dots, 1 \le i_d \le p_d} f_T^M(I_1^{i_1} I_2^{i_2} \dots I_d^{i_d}), \end{aligned} \tag{7.19}$$

where $p_1, p_2, \dots, p_d \le 2(b-1)\log_b m$.

Multi-dimensional HI Mechanism $(\mathcal{A}_{\mathsf{HI}}, \mathsf{P}_{\mathsf{HI}})$

On the client side, an augmented dimension tells which d-dim interval a user belongs to. We use $\mathcal{A}_{\mathsf{FO}}$ in a weighted frequency oracle to encode all the $(h+1)^d$ augmented dimensions, each of which uses a privacy budget of $\epsilon/(h+1)^d$. On the server side, from how q is rewritten in (7.19), we can estimate the weighted frequency of each d-dim interval and sum up the estimates to approximate the answer to q:

$$\mathsf{P}_{\mathsf{HI}}(q) = \sum_{1 \le i_1 \le p_1, \dots, 1 \le i_d \le p_d} \hat{f}_T^M(I_1^{i_1} I_2^{i_2} \dots I_d^{i_d}). \tag{7.20}$$

Theorem 7.4.1 (HI)**.** *i)* $\mathcal{A}_{\mathsf{HI}}$ *satisfies* ϵ*-LDP. ii)* $\mathsf{P}_{\mathsf{HI}}(q)$ *is an unbiased estimator of* q *with expected squared error*

$$\begin{aligned} \mathrm{Err}(\mathsf{P}_{\mathsf{HI}}(q)) &\le (2(b-1)\log_b m)^{d_q} M_T^2 \cdot \frac{(e^{\epsilon/(\log_b m+1)^d}+1)^2}{(e^{\epsilon/(\log_b m+1)^d}-1)^2} \\ &= \mathrm{O}\left(\frac{n\Delta^2 \log^{d_q+2d} m}{\epsilon^2}\right) \ \textit{when } \epsilon \textit{ is small}, \end{aligned} \tag{7.21}$$

where $n = |T|$ *is the number of users,* d *(*d_q*) is the number of sensitive dimensions (in the query* q*),* Δ *is the range of* M*,* $M_T^2 = \sum_{t \in T} t[M]^2$*, and the constant* b *is the fan-out.*

Algorithm 4 d-dim HI Optimized $(\mathcal{A}_{\mathsf{HIO}}, \mathsf{P}_{\mathsf{HIO}})$

Client side: Encode dimensions $t[D_1], \ldots, t[D_d]$.

1: Randomly pick $(l_1, \ldots, l_d) \in \{0, 1, \ldots, h\}^d$.

2: Suppose $t[D_i]$ is in interval $I_i^{l_i} \in \mathcal{L}_{D_i}^{l_i}$ $(i = 1, \ldots, d)$; let $t[\mathcal{L}_{D_1}^{l_1} \times \ldots \times \mathcal{L}_{D_d}^{l_d}] \leftarrow I_1^{l_1} I_2^{l_2} \ldots I_d^{l_d}$.

3: Create LDP report:

$$\mathcal{A}_{\mathsf{HIO}}(t) \leftarrow \left((l_1, \ldots, l_d), \mathcal{A}_{\mathsf{FO}}^{\epsilon}(t[\mathcal{L}_{D_1}^{l_1} \times \ldots \times \mathcal{L}_{D_d}^{l_d}])\right). \tag{7.22}$$

Server side: MDA query $q = Q_T(\mathsf{SUM}(M), D_1 \in [l_1, r_1] \wedge \ldots \wedge D_d \in [l_d, r_d])$.

1: Let $S_{(l_1,\ldots,l_d)} \leftarrow \emptyset$ for each $(l_1, \ldots, l_d) \in \{0, 1, \ldots, h\}^d$.

2: For each user t, if we get $((l_1, \ldots, l_d), \mathcal{A}_{\mathsf{FO}}^{\epsilon}(\cdot))$: $S_{(l_1,\ldots,l_d)} \leftarrow S_{(l_1,\ldots,l_d)} + \{t\}$;

3: For $i = 1$ to d do:

4: Decompose $[l_i, r_i]$ into p_i disjoint intervals $[l_i, r_i] \rightarrow$

5: $I_i^1 \cup I_i^2 \cup \ldots \cup I_i^{p_i}$ in the hierarchy $\mathbf{I}_{D_i}$;

6: For each $(i_1, \ldots, i_d) \in \{1, \ldots, p_1\} \times \ldots \times \{1, \ldots, p_d\}$:

7: Estimate $f_T^M(I_1^{i_1} I_2^{i_2} \ldots I_d^{i_d})$ as (suppose $I_k^{i_k} \in \mathcal{L}_{D_k}^{l_k}$):

$$\tilde{f}_{T,1/(h+1)^d}^M(I_1^{i_1} I_2^{i_2} \ldots I_d^{i_d}) = (h+1)^d \cdot \hat{f}_{S_{(l_1,\ldots,l_d)}}^M(I_1^{i_1} I_2^{i_2} \ldots I_d^{i_d}); \tag{7.23}$$

8: Output an estimation to q as:

$$\mathsf{P}_{\mathsf{HIO}}(q) = \sum_{1 \le i_1 \le p_1, \ldots, 1 \le i_d \le p_d} \tilde{f}_{T,1/(h+1)^d}^M(I_1^{i_1} I_2^{i_2} \ldots I_d^{i_d}). \tag{7.24}$$

Boosting Accuracy via User Partitioning

Similar to the 1-dim case in Section 7.3.2, HI's accuracy can be boosted by randomly partitioning users by levels. On the client, a user picks one of the $(h+1)^d$ d-dim levels randomly, and encodes only the d-dim interval in this level with privacy budget ϵ. On the server, we estimate the weighted frequency $f_T^M(I_1^{i_1} I_2^{i_2} \ldots I_d^{i_d})$ in (7.19) with LDP reports in the corresponding level from a random $1/(h+1)^d$ portion of users (as in Section 7.2.2).

The resulting mechanism $(\mathcal{A}_{\mathsf{HIO}}, \mathsf{P}_{\mathsf{HIO}})$ is in Algorithm 4. Theorem 7.4.2 shows that the gain from a larger privacy budget spent on the picked level per user overcomes the error due to running weighted frequency oracles on samples. $\mathsf{P}_{\mathsf{HIO}}$ has a significant accuracy boost over P_{HI}.

Theorem 7.4.2 (HIO). *i)* $\mathcal{A}_{\mathsf{HIO}}$ *satisfies* ϵ*-LDP. ii)* $\mathsf{P}_{\mathsf{HIO}}(q)$ *is an unbiased estimator of* q *with expected squared error*

$$\begin{aligned}&\mathrm{Err}(\mathsf{P}_{\mathsf{HIO}}(q)) \\ \leq&(2(b-1)(\log_b m+1))^{d_q}(\log_b m+1)^d M_T^2 \cdot \frac{(e^{2\epsilon}+1)}{(e^\epsilon-1)^2} \\ =&\mathrm{O}\left(\frac{n\Delta^2 \log^{d_q+d} m}{\epsilon^2}\right) \text{ when } \epsilon \text{ is small,}\end{aligned} \tag{7.25}$$

where $n = |T|$ *is the number of users,* d *(*d_q*) is the number of private dimensions (in the query* q*),* Δ *is the range of* M*,* $M_T^2 = \sum_{t\in T} t[M]^2$*, and the constant* b *is the fan-out.*

7.4.2 Ordinal and Categorical Dimensions

A categorical dimension D can be regarded as a hierarchy with two levels: $\mathcal{L}_D^0 = \{*\}$ and $\mathcal{L}_D^1 = \{[v_1], [v_2], \ldots, [v_c]\}$, where '$*$' means 'anything', and $v_1, v_2, \ldots, v_c$ are distinct values D. As there are only point constraints on D, e.g., "$D = v_i$", all the intermediate levels are unnecessary. Such a *categorical hierarchy* can be incorporated into the multi-dimensional hierarchy of intervals introduced in Section 7.4.1.

7.4.3 Split-and-Conjunction: When the Dimensionality is High

HI and HIO mechanisms have errors exponentially depending on both the number of dimensions in the query (d_q) and the total number of sensitive dimensions in the data model (d). On the client side, they partition privacy budget ϵ or users into $\Theta(\log^d m)$ portions, one for each d-dim level, and thus may introduce too much LDP noise for large d. When $d_q \ll d$, there is room for improvement: whether it is possible to remove the exponential dependency on d.

We introduce our *split-and-conjunction (SC) mechanism* in this section. Instead of encoding a tuple on the d-dim hierarchy, a user maintains d one-dim hierarchies, on which the d dimensions are encoded and reported independently. The privacy budget ϵ is thus partitioned into $\Theta(d \log m)$ portions (d dimensions each with $\Theta(\log m)$ one-dim levels). The

question is, while all dimensions are reported independently, whether we can estimate how many rows have, e.g., $t[D_1] = v_1$ *and* $t[D_2] = v_2$ conjunctively. To this end, we will first introduce a new class of estimators in frequency oracles, called *conjunctive estimators* as a building block of SC.

Conjunctive Estimators $\bar{f}$ and $\bar{f}^M$

Let D_1 and D_2 be two sensitive dimensions. A *conjunctive weighted frequency query* asks, for a set of users S, and $v_1 \in D_1$ and $v_2 \in D_2$, the total measure of users with $t[D_1] = v_1$ and $t[D_2] = v_2$, i.e., weighted frequency $f_S^M(v_1v_2) = \sum_{t \in S \wedge t[D_1]=v_1 \wedge t[D_2]=v_2} t[M]$ of $\langle v_1, v_2 \rangle$ on the domain $D_1 \times D_2$, or unweighted frequency

$$f_S(v_1v_2) = |\{t \in S \mid t[D_1] = v_1 \wedge t[D_2] = v_2\}|.$$

Given two sets of independently-generated LDP reports $\{\mathcal{A}_{\mathsf{FO}}(t[D_1])\}_{t \in S}$, $\{\mathcal{A}_{\mathsf{FO}}(t[D_2])\}_{t \in S}$, we want to estimate $f_S^M(v_1v_2)$, for any $v_1 \in D_1$ and $v_2 \in D_2$. More generally, we can estimate $f_S^M(v_1 \dots v_k)$ from reports on k dimensions. For this purpose, we introduce *conjunctive estimators* $\bar{f}$ and $\bar{f}^M$.

States and transition. Recall that an LDP report $\mathcal{A}_{\mathsf{FO}}(t[D_i]) = \langle H, y \rangle$ is a pair of a random hash function H and a randomly perturbed value y (refer to Section 3.3.5).

We define two indicator variables for the query term v_i:

$$\textit{input state}: \quad B_i(t) = \begin{cases} 0, & \text{if } \ t[D_i] \neq v_i \\ 1, & \text{if } \ t[D_i] = v_i \end{cases} \quad \text{and} \tag{7.26}$$

$$\textit{output state}: \quad A_i(t) = \begin{cases} 0, & \text{if } \ H(v_i) \neq y \\ 1, & \text{if } \ H(v_i) = y \end{cases} \quad (\text{for } i = 1, 2). \tag{7.27}$$

Input state $B_i(t)$ is deterministic and depends on v_i; output state $A_i(t)$ is a random variable and depends on both v_i and randomness in $\mathcal{A}_{\mathsf{FO}}$. We can define the following *transition probabilities*:

$$P_{b\to a} \triangleq \mathsf{Pr}\left[A_i(t) = a \mid B_i(t) = b\right] \quad (\text{for } a, b \in \{0, 1\}).$$

A tuple with two dimensions has 4 possible 2-*dim input states*: "11" (meaning $B_1(t) = 1 \wedge B_2(t) = 1$), "01", "10", and "00". After applying $\mathcal{A}_{\mathsf{FO}}$, the LDP report has 4 possible 2-*dim output states* "11" (meaning $A_1(t) = 1 \wedge A_2(t) = 1$), "01", "10", and "00". We can derive 2-*dim transition probabilities*:

$$\begin{aligned}
&P_{b_1b_2\to a_1a_2} \quad (\text{for } a_1, a_2, b_1, b_2 \in \{0, 1\}) \\
&\triangleq \mathsf{Pr}\left[A_1(t) = a_1 \wedge A_2(t) = a_2 \mid B_1(t) = b_1 \wedge B_2(t) = b_2\right] \\
&= P_{b_1\to a_1} \cdot P_{b_2\to a_2} \text{ (as } A_1(t) \text{ and } A_2(t) \text{ are independent)},
\end{aligned}$$

since dimensions are encoded independently.

Estimation via transition matrix. For a set S of users and their reports $\{\mathcal{A}_{\mathsf{FO}}(t[D_1])\}_{t\in S}$ and $\{\mathcal{A}_{\mathsf{FO}}(t[D_2])\}_{t\in S}$, we observe the frequency of each 2-dim output state. With them, we can estimate the frequencies of input states via transition probabilities, and in particular, the frequency of 2-dim input state "11" is equal to unweighted frequency $f_S(v_1v_2)$.

The frequencies of input states are, for $b_1b_2 = 11, 01, 10, 00$:

$$\mathbf{b}_S(b_1b_2) = |\{t \in S \mid B_1(t) = b_1 \wedge B_2(t) = b_2\}|.$$

And those of output states are, for $a_1a_2 = 11, 01, 10, 00$:

$$\mathbf{a}_S(a_1a_2) = |\{t \in S \mid A_1(t) = a_1 \wedge A_2(t) = a_2\}|.$$

Consider the corresponding frequency vectors of input and output states, $\mathbf{b} = [\mathbf{b}_S(11), \mathbf{b}_S(01), \mathbf{b}_S(10), \mathbf{b}_S(00)]^\top$ and $\mathbf{a} = [\mathbf{a}_S(11), \mathbf{a}_S(01), \mathbf{a}_S(10), \mathbf{a}_S(00)]^\top$. We can establish the relationship between $\mathbf{b}$ and $\mathbf{a}$ through transition matrix $\mathbf{P}$:

$$\mathbf{P} \cdot \mathbf{b} \triangleq \begin{pmatrix} P_{11\to 11} & P_{01\to 11} & P_{10\to 11} & P_{00\to 11} \\ P_{11\to 01} & P_{01\to 01} & P_{10\to 01} & P_{00\to 01} \\ P_{11\to 10} & P_{01\to 10} & P_{10\to 10} & P_{00\to 10} \\ P_{11\to 00} & P_{01\to 00} & P_{10\to 00} & P_{00\to 00} \end{pmatrix} \begin{pmatrix} \mathbf{b}_S(11) \\ \mathbf{b}_S(01) \\ \mathbf{b}_S(10) \\ \mathbf{b}_S(00) \end{pmatrix} = \mathrm{E}\left[\, \mathbf{a} \,\right],$$

from the property of $\mathbf{P}$ and the linearity of expectation. By observing the frequency vector $\mathbf{a}$, we can estimate $\mathbf{b}$ as:

$$\hat{\mathbf{b}} = [\hat{\mathbf{b}}_S(11), \hat{\mathbf{b}}_S(01), \hat{\mathbf{b}}_S(10), \hat{\mathbf{b}}_S(00)]^\top = \mathbf{P}^{-1} \cdot \mathbf{a} \tag{7.28}$$

$$\text{and, in particular,} \quad \bar{f}_S(v_1 v_2) = \hat{\mathbf{b}}_S(11). \tag{7.29}$$

From the linearity of expectation, we have $\mathrm{E}\left[\, \hat{\mathbf{b}} \,\right] = \mathbf{P}^{-1} \cdot \mathrm{E}\left[\, \mathbf{a} \,\right] = \mathbf{P}^{-1}\mathbf{P} \cdot \mathbf{b} = \mathbf{b}$, and thus $\mathrm{E}\left[\, \bar{f}_S(v_1 v_2) \,\right] = \mathbf{b}_S(11) = f_S(v_1 v_2)$.

Similar to (7.7), for the weighted case, we can derive

$$\bar{f}_S^M(v_1 v_2) = \sum_{\text{distinct } x} x \cdot \bar{f}_{S_x}(v_1 v_2), \tag{7.30}$$

where $S_x = \{t \in S \mid t[M] = x\}$. Its unbiasedness is from $\bar{f}$'s.

In order to extend $\bar{f}$ and $\bar{f}^M$ from two dimensions to d dimensions, we extend 2-dim input/output states to d-dim input/output states. Correspondingly, their frequencies are:

$$\mathbf{b}_S(b_1 \dots b_d) = |\{t \in S \mid B_1(t) = b_1 \wedge \dots \wedge B_d(t) = b_d\}|,$$
$$\mathbf{a}_S(a_1 \dots a_d) = |\{t \in S \mid A_1(t) = a_1 \wedge \dots \wedge A_d(t) = a_d\}|.$$

The transition matrix $\mathbf{P}$ is a $2^d \times 2^d$ one, as there are 2^d states. The errors in $\bar{f}$ and $\bar{f}^M$ can be bounded as follows.

Proposition 7.4.1 (Conjunction of Oracles). *Run an ϵ-LDP encoder $\mathcal{A}^{\epsilon}_{\mathsf{FO}}$ on each of the d dimensions independently (overall, the procedure is $(d\epsilon)$-LDP). For any k dimensions and values $v_1, \ldots, v_k$ on them, we have unbiased estimators $\bar{f}_S$ and $\bar{f}^M_S$ of conjunctive unweighted and weighted frequencies, respectively,*

$$\begin{aligned} \textit{with error} \quad & \mathrm{Err}(\bar{f}_S(v_1 \ldots v_k)) = \mathrm{O}\Big(|S|/\epsilon^{2k}\Big) \quad \textit{and} \\ & \mathrm{Err}(\bar{f}^M_S(v_1 \ldots v_k)) = \mathrm{O}\Big(|S|\Delta^2/\epsilon^{2k}\Big), \end{aligned}$$

where $\Delta = \max(M) - \min(M)$. If we guarantee ϵ-LDP across all the d dimensions, each dimension gets a privacy budget ϵ/d, and thus, the error of $\bar{f}^M_S$ is $\mathrm{O}\Big(|S|\Delta^2/(\epsilon/d)^{2k}\Big)$.

Split-and-Conjunction (SC) Mechanism

We now describe our SC mechanism $(\mathcal{A}_{\mathsf{SC}}, \mathsf{P}_{\mathsf{SC}})$. On the client side, a user reports each one-dim interval s/he belongs to (one per level) in each one-dim hierarchy $\mathbf{I}_{D_i}$ using $\mathcal{A}_{\mathsf{FO}}$ independently, with a privacy budget of $\epsilon/(dh)$. The estimator on the server side is the same as the one in HI mechanism, except that, instead of $\hat{f}^M$, the conjunctive estimator $\bar{f}^M$ (from Section 7.4.3) is used as we have no access to the (LDP version of) d-dim hierarchy. Error in the estimated answer to an MDA query is exponentially dependent on only d_q (number of private dimensions in q).

Theorem 7.4.3 (SC). *i) $\mathcal{A}_{\mathsf{SC}}$ satisfies ϵ-LDP. ii) $\mathsf{P}_{\mathsf{SC}}(q)$ is an unbiased estimator of q with expected squared error*

$$\mathrm{Err}(\mathsf{P}_{\mathsf{SC}}(q)) = \mathrm{O}\left(\frac{n\Delta^2 d^{2d_q} \log^{3d_q} m}{\epsilon^{2d_q}}\right) \quad \textit{when } \epsilon \textit{ is small}, \tag{7.31}$$

where $n = |T|$ is the number of users, d (d_q) is the number of private dimensions (in the query q), and Δ is the range of M.

7.4.4 Performance Comparison

While the accuracy of mechanisms introduced so far depends on some common parameters, e.g., ϵ and data size, they depends on others, e.g., number/sizes of dimensions, in different ways. We try to identify the analytical turning points of their performance (the best ones in different settings). These analytical results will be verified in our experiments later.

Marginal/FO v.s. HIO. Worst-case errors in the marginal or FO-based solution (introduced in Section 7.2.3) depend on ϵ, $|T|$, and Δ in the same way as errors in HIO asymptotically.

Consider the marginal with all the dimensions in the predicate. Define the *volume* $\text{vol}(q)$ of a query q to be the ratio of marginal rows satisfying its predicate to all marginal rows. A 1-dim query with a range constraint $D \in [l, r]$ has volume $\text{vol}(q) = (r-l+1)/m$, From (7.10) in Section 7.2.3 and Theorem 7.3.2, HIO is better than the marginal/FO-based solution if

$$(r - l + 1) \geq \Theta(\log^2 m) \Leftrightarrow \text{vol}(q) \geq \Theta(\log^2 m/m). \tag{7.32}$$

If there are d sensitive dimensions, from Section 7.2.3 and Theorem 7.4.2, HIO is better than the marginal-based solution if

$$\text{vol}(q) \geq \Theta(\log^{2d} m/m^d) \quad (\text{when } d_q = d). \tag{7.33}$$

HIO v.s. SC. Comparing Theorem 7.4.3 to Theorem 7.4.2, SC removes d from the power of $\log m$ in the error, but incurs an additional term d^{2d_q}. Only when (from Theorems 7.4.2 and 7.4.3)

$$(d \log m/\epsilon)^{2d_q} \leq \Theta(\log^d m/\epsilon^2), \tag{7.34}$$

i.e., d_q is small enough relative to d, SC is better than HIO.

7.5 Evaluation

We evaluate our mechanisms in various settings. In short, HIO performs the best most of time, with a normalized absolute error less than 5% in most queries, and a relative error less than 5% if the predicate is not too selective; SC performs better in high-dimensional settings if the number of dimensions in the query is much less than the total number of dimensions. We also conduct a case study in a big-data platform. Mechanisms are implemented in Python and evaluated on an Intel Xeon E5 2682 v4 PC with 64GB memory.

Datasets. We conduct experiments on three datasets:

- Adult [37]: A dataset from the UCI ML repository [33] with around 45 thousand tuples after removing missing values.
- IPUMS [34]: A 2016 US census dataset from the IPUMS repository [33]. It contains around 3 million records.
- A real dataset with 150 million records from an e-commerce application. Details are deferred to Section 7.5.2.

Experiment settings. We compare the four mechanisms:

- MG: Processing MDA queries with the state-of-the-art LDP marginal-releasing technique [24] (described in Section 7.2.3).
- HI: Hierarchical-interval mechanism $(\mathcal{A}_{\mathsf{HI}}, \mathsf{P}_{\mathsf{HI}})$.
- HIO: HI Optimized $(\mathcal{A}_{\mathsf{HIO}}, \mathsf{P}_{\mathsf{HIO}})$.
- SC: Split-and-conjunction mechanism $(\mathcal{A}_{\mathsf{SC}}, \mathsf{P}_{\mathsf{SC}})$.

We use fan-out $b = 5$ for HI, HIO, and SC; and we test SUM/COUNT/AVG queries (SUM by default).

Error measures. We use two error measures:

- Mean Normalized Absolute Error MNAE = $\mathsf{AVG}_q(\frac{|\mathsf{P}(q)-q|}{\Sigma_S})$, where the absolute error is normalized by $\Sigma_S = \sum_t |t[M]|$ to $[0, 1]$. It is used for SUM queries, and measures how large errors are relative to the maximum possible answer (Σ_S).

- Mean Relative Error MRE = $\mathsf{AVG}_q(\frac{|\mathsf{P}(q)-q|}{|\mathsf{P}(q)|})$. It is used for SUM/COUNT/AVG queries, and measures how large the error is relative to the true answer for each query.

In Section 7.5.1, we use MNAE for SUM queries to verify theoretical results and compare the mechanisms, as they all have theoretical guarantees about absolute errors, which are independent of (or not dominated by) query sizes/answers. MRE is a query-dependent metric: for queries with very small, e.g., 0, (or very large) true answers, MRE is very large, e.g., infinity, (or very small) in all the mechanisms, and thus sometimes barely gives a clear distinction of their accuracy.

In Section 7.5.2, we use relative error MRE to demonstrate the utility of our mechanisms for SUM/COUNT/AVG queries. Queries are partitioned into groups by selectivity of their predicates: in each group, queries have similar sizes and answers, and thus MRE's for different queries are comparable. MRE is also reported in a case study to demonstrate the utility of our mechanism in a real-world deployment.

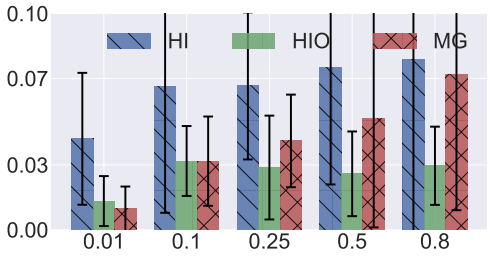
For each data point in every figure, we test 30 random queries and plot their MNAE or MRE (Y-axis) with 1-std.

7.5.1 Experimental Comparison

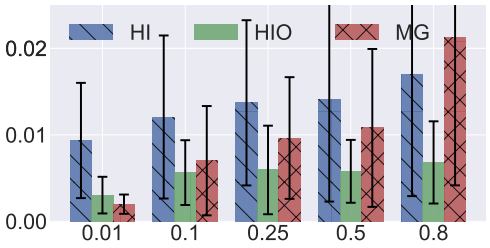
We compare different mechanisms for varying factors that potentially influence the accuracy: i) query volume vol(q) (defined in Section 7.4.4), ii) number of dimensions, iii) domain sizes (cardinalities of dimensions), iv) data size, and v) ϵ.

One Ordinal Dimension

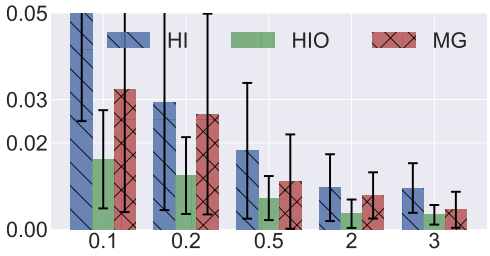
We start from MDA queries with one sensitive ordinal dimension: $q = Q_T(\mathsf{SUM}(M), D \in [l, r])$. We create the ordinal dimension with size $m = 1024$ by bucketizing a numeric column of the table.



(a) Adult: vary vol(q)



(b) IPUMS 1M: vary vol(q)



(c) IPUMS: vary $|T|$ (million)

Figure 7.3. Comparing different mechanisms: vary query volume and data size ($\epsilon = 2$ and $d = 1$)

Varying query volume. Figures 7.3a-7.3b show the MNAE for different query volumes on Adult and IPUMS (1M sample). The interval $[l, r]$ is generated randomly with $r - l + 1 = m \cdot \text{vol}(q)$. The accuracy of MG deteriorates quickly as vol(q) increases. Only when vol(q) is as small as 0.01, MG is better than HIO (the errors of both are small, too); vol(q) = 0.1 is the break-even point, which conforms with our analysis in Section 7.4.4; when vol(q) = 0.8, error of MG is $\sim 3\times$ that of HIO. The performance of different methods on IPUMS is better than that on Adult, because the data size is larger.

Varying data size. We sample $0.1, 0.2, 0.5, 2$ and 3 million rows from IPUMS (without replacement), and test queries with volume 0.25. As can be seen in Figure 7.3c, the larger the dataset, the better the estimation accuracy, which is consistent with our theorems. HIO always performs best.

Varying privacy budget ϵ. Figure 7.4 shows performance of different mechanisms on IPUMS for varying ϵ. All methods benefit from larger ϵ, with HIO performing the best.

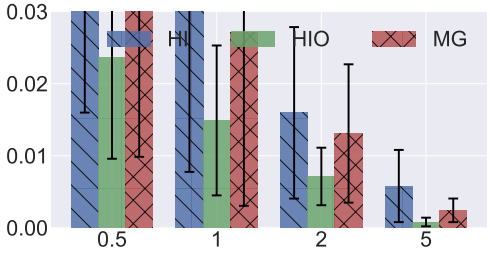
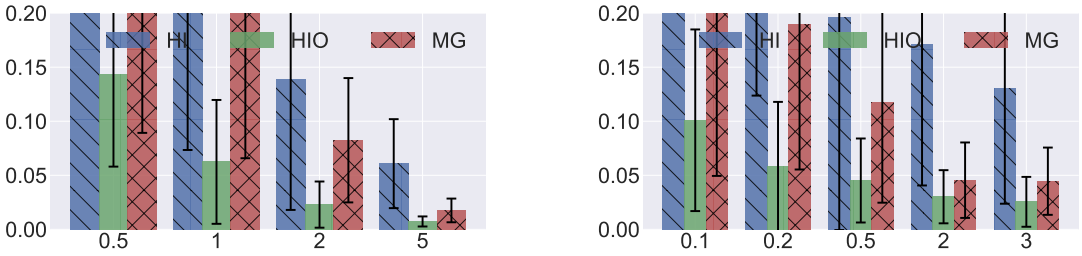


Figure 7.4. IPUMS 1M: vary ϵ ($d = 1$)



(a) $m_1 \times m_2 = 256 \times 256$: vary ϵ (b) $m_1 \times m_2 = 256 \times 256$: vary $|T|$ (million)

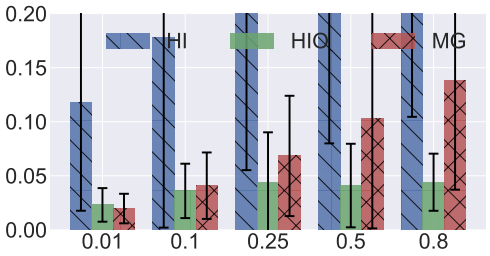
Figure 7.5. Two sensitive ordinal dimensions: vary ϵ and data size ($d = 2$)

In the rest of Section 7.5.1, we will use $\text{vol}(q) = 0.25$, data size $|T| = 1$ million, and $\epsilon = 2$, as their default values.

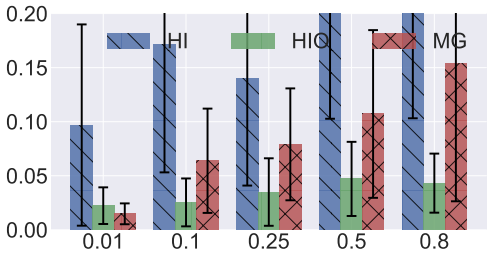
Two Ordinal Dimensions

We create two ordinal dimensions on IPUMS with sizes m_1 and m_2 by bucketizing numeric columns. Two configurations of $m_1 \times m_2$ are tested: 256×256 and 1024×64. The query predicate is the conjunction of two range constraints: $D_1 \in [l_1, r_1] \wedge D_2 \in [l_2, r_2]$, with volume $\text{vol}(q) = (r_1 - \ell_1 + 1) \times (r_2 - \ell_2 + 1)/(m_1 \times m_2)$.

Figures 7.5-7.6 shows the results of different mechanisms for the two configurations, when varying ϵ, $|T|$, and $\text{vol}(q)$. When $\text{vol}(q) \leq 0.01$, MG is better, which is consistent with our analysis in Section 7.4.4; otherwise, MG is much worse than HIO for varying ϵ and $|T|$, as an MDA query with two range constraints can be decomposed into too many marginal cells, whose errors accumulate when being aggregated.



(a) $m_1 \times m_2 = 256 \times 256$: vary vol($q$)



(b) $m_1 \times m_2 = 1024 \times 64$: vary vol($q$)

Figure 7.6. Two sensitive dimensions: vary query volume ($\epsilon = 2$ and $d = 2$)

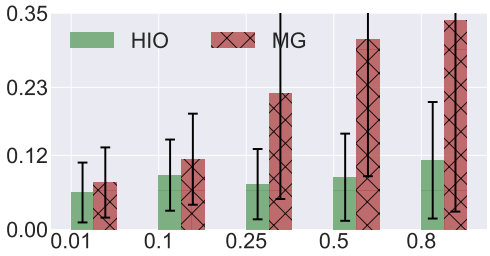


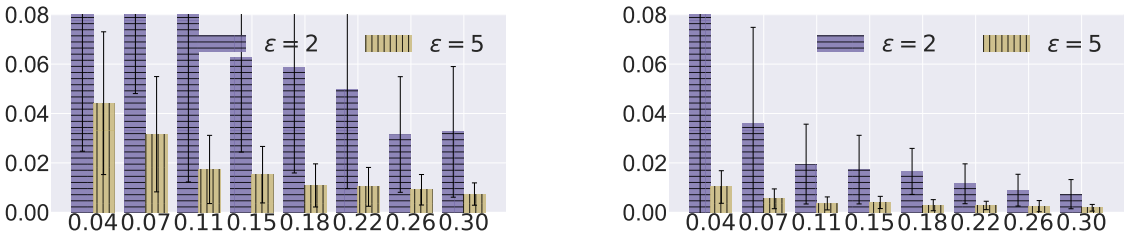
Figure 7.7. Three sensitive dimensions: vary query volume ($\epsilon = 2$ and $d = 3$)

Table 7.2. One-run estimations (using HIO) of sample AVG queries and true answers

$\epsilon =$	0.5	1	2	5	true
$Q1$	26.29	24.36	26.81	26.07	26.32
$Q2$	36.97	32.36	33.77	32.79	33.11
$Q3$	27.07	34.69	24.22	26.68	27.01

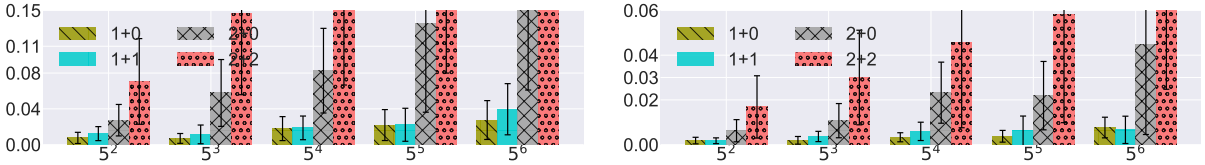
Three Ordinal Dimensions

We create three sensitive ordinal dimensions on IPUMS with sizes $256 \times 256 \times 64$. The query predicate is the conjunction of the two range constraints. Errors in HI are much larger than those in HIO, so we omit it here. Figure 7.7 shows the results of HIO and MG, when varying vol(q). Errors in MG highly depend on vol(q), which is consistent with our analysis in Sections 7.2.3 and 7.4.4. On the other hand, although more challenging, HIO can still work, with consistently better estimations than MG. When vol(q) ≥ 0.5, HIO is at least $3\times$ more accurate.



(a) SUM queries (b) AVG queries

Figure 7.8. Relative error of HIO: vary selectivity



(a) $\epsilon = 2$: vary domain size (b) $\epsilon = 5$: vary domain size

Figure 7.9. Relative error of HIO on 2 (ordinal) + 2 (categorical) dimensions: vary domain sizes and query types (SUM queries)

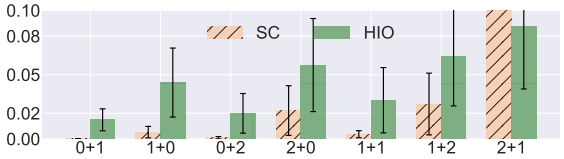


Figure 7.10. Relative error of HIO and SC on 4 (ordinal) + 4 (categorical) dimensions: vary query types ($\epsilon = 5$)

7.5.2 Relative Error and Practical Usage

We now focus on the utility of our mechanisms in more real settings and conduct a case study. We will report their relative errors (MRE). We consider data models with four or more sensitive (both ordinal and categorical) dimensions, but MDA queries may or may not contain all of them. When there are more than two sensitive dimensions, HI and MG already give much worse accuracy than HIO does. Thus, their performances are not reported here. We will evaluate HIO and SC, as their benefit is more pronounced when there are more sensitive attributes (analyzed in Section 7.4.4).

Two Ordinal and Two Categorical Dimensions

We start with four sensitive dimensions in IPUMS. We let both ordinary dimensions have the same domain size m (5^4 by default). We can change m through finer or looser bucketization.

Sample queries on with varying ϵ. Three sample queries $Q1$-$Q3$ are processed using HIO, and their estimated/true answers shown in Table 7.2. The relative errors of estimations are within 5% most of the time. $Q3$ has the most selective predicate, and error in its estimated answers is also the largest among the three.

Varying selectivity of predicate. Since our theorems give guarantees on absolute errors for COUNT and SUM queries (e.g., Theorem 7.4.2), relative errors are highly impacted by sizes of query answers, which in-turn depend on the selectivities of predicates. We test COUNT, SUM, and AVG queries with four types of predicates, 1+0, 1+1, 2+0, and 2+2 (*a+b means a ordinal dimensions and b categorical dimensions*), and varying selectivities. We plot the results in Figure 7.8 (COUNT queries have very similar trends to SUM). Their relative error decreases with increasing selectivity. AVG is estimated as SUM/COUNT, and thus, its relative error has a similar trend. As long as predicates are not too selective, we get reasonable relative errors for both $\epsilon = 2$ and 5.

Varying query type and domain size. For varying domain sizes and different types of predicates, we evaluate HIO and SC on queries with selectivity around 0.1. SC is worse in almost all types. Errors of HIO are reported in Figure 7.9. The errors are larger when the domains are larger due to the $\log m$ term in the error bounds. Also, queries of types 1+0 and 1+1 can be answered more accurately than 2+0 and 2+2 queries, which is consistent with Theorems 7.4.2 and 7.4.3 – the error increases as d_q increases.

Four Ordinal and Four Categorical Dimensions

We evaluate HIO and SC in the 8-dim setting, for different query types, and report the results in Figure 7.10. In this high-dim setting, according to Section 7.4.4, SC should give better estimations for queries with fewer dimensions in the predicates. Empirically, SC

Table 7.3. One-run estimated answers in the case study

$\epsilon =$	0.5	1	2	5	true	selectivity
$Q4$	0.185	0.154	0.178	0.167	0.168	0.049
rel. err.	0.102	0.081	0.061	0.005	-	-
$Q5$	0.157	0.148	0.160	0.170	0.171	0.011
rel. err.	0.086	0.138	0.066	0.008	-	-

performs better than HIO for the almost all the query types except 2+1. This is consistent with our analytical results and is the purpose of introducing SC.

Case Study: E-Commerce Analytics

We test HIO for a real-world application in an e-commerce company, who wants to collect delivery information from users in a privacy-preserving way. The table T collected via HIO contains more than 150 million users with four attributes. Attributes about each user's location (Region) and the product s/he bought (Category, Price) are sensitive dimensions, and the postage fee (Postage) is a public measure attribute. Suppose we want to analyze the postage distribution for certain group of users and products, Table 7.3 gives the answers estimated by HIO for different ϵ. With the a large number of users, the accuracy is much improved.

7.6 Extensions and Discussion

Other space partitioning techniques. Frequency oracles can be combined with QuadTree to handle MDA queries: intuitively, a user can encode the tree nodes containing her/his tuple locally via frequency oracles. However, QuadTree incurs larger errors, because, to answer a 2D range query, in the worst case, the entire QuadTree needs to be traversed and thus too many noisy counts (the number is linear in the domain size) are added up which amplifies the error.

Coefficients in wavelet transforms (used in Privelet [38]) can be encoded using frequency oracles. Each user randomly selects a level in the decomposition tree of the wavelet transform,

and reports his location on that level. However, as each level has a different weight in the estimation, it is unclear how to partition users across levels to optimize the utility.

In general, the idea proposed in this paper can also be used in other space partitioning techniques. Basically, each user has a local view of the data structure (with one data point), and report this structure under LDP. This can be optimized by having each user randomly select a sub-structure that has a fixed sensitivity (e.g., a level). The server collects and adds up the LDP data structure from users, and then uses the result to answer queries. It is interesting future work to customize frequency oracles for different space indexing techniques and compare them systematically.

Non-sensitive + private dimensions in predicates. If an MDA query has both private and public dimensions in the predicate, the server can evaluate the public part first, and, process the remaining rows/users with estimation processor P in our mechanisms. For example, in Table 7.1, the query

$$
\begin{aligned}
&\mathsf{SELECT}\ \mathsf{SUM}(\mathsf{Purchase})\ \mathsf{FROM}\ T\ \mathsf{WHERE}\\
&\mathsf{Age} \in [30, 40]\ \mathsf{AND}\ \mathsf{OS} = \mathrm{Win};\quad \Leftrightarrow\quad \text{can be evaluated as:}\\
&T_{\mathrm{pub}} = \mathsf{SELECT}\ *\ \mathsf{FROM}\ T\ \mathsf{WHERE}\ \mathsf{OS} = \mathrm{Win};\\
&\mathsf{SELECT}\ \mathsf{SUM}(\mathsf{Purchase})\ \mathsf{FROM}\ T_{\mathrm{pub}}\ \mathsf{WHERE}\ \mathsf{Age} \in [30, 40];
\end{aligned}
$$

where T_{pub} can be evaluated using a normal query processor, and the last line is processed with P in our mechanisms.

Handling other aggregation functions. As long as the aggregation function can be rewritten as $\mathsf{SUM}()$ functions, our mechanisms can be extended to handle it. For example, $\mathsf{STDEV}(M)$ can be computed from $\mathsf{SUM}(M^2)$, $\mathsf{SUM}(M)$, and COUNT. We can also support aggregations on multiple measures, e.g., $\mathsf{SUM}(a \cdot M_1 + b \cdot M_2)$, as long as M_1 and M_2 are public (conceptually, define $M = a \cdot M_1 + b \cdot M_2$).

AND-OR expressions. An MDA query with OR in the predicate can be rewritten as sub-queries with only AND using the inclusion-exclusion principle. For example,

$$\text{SELECT SUM(Purchase) FROM } T$$
$$\text{WHERE Age} \in [30, 40] \text{ OR Salary} \in [50\text{K}, 150\text{K}]$$

can be rewritten as three sub-queries: "... Age $\in [30, 40]$" + "... Salary $\in [50\text{K}, 150\text{K}]$" − "... Age $\in [30, 40]$ AND Salary $\in [50\text{K}, 150\text{K}]$", each to be estimated using P in our mechanisms. More generally, an AND-OR expression can be converted into a disjunctive normal form, and we apply the inclusion-exclusion principle over clauses as in the above example.

8. POST PROCESSING

(A version of this chapter has been previously published in NDSS 2020 [39].)

While existing state-of-the-art frequency oracles developed in Chapter 3 are designed to provide unbiased estimations while minimizing the variance, it is possible to further reduce the variance by performing post-processing steps that use prior knowledge to adjust the estimations. In particular, there are two consistency requirements for frequency:

1. The estimated frequency of each value is non-negative.

2. The sum of the estimated frequencies is 1.

For example, exploiting the property that all frequency counts are non-negative can reduce the variance; however, simply turning all negative estimations to 0 introduces a systematic positive bias in all estimations. By also ensuring the property that the sum of all estimations must add up to 1, one ensures that the sum of the biases for all estimations is 0. However, even though the biases cancel out when summing over the whole domain, they still exist. There are different post-processing methods that were explicitly proposed or implicitly used. They will result in different combinations of variance reduction and bias distribution. Selecting a post-processing method is similar to considering the bias-variance tradeoff in selecting a machine learning algorithm.

8.1 Towards Consistent Frequency Oracles

We study the property of several post-processing methods, aiming to understand how they compare under different settings, and how they relate to each other. Our goal is to identify efficient post-processing methods that can give accurate estimations for a wide variety of queries. We first present the baseline method that does not do any post-processing.

- **Base**: *We use the standard* FO *to obtain estimations of each value.*

Base has no bias, and its variance can be analytically computed (e.g., using [5]).

8.1.1 Baseline Methods

When the domain is large, there will be many values in the domain that have a zero or very low true frequency; the estimation of them may be negative. To overcome negativity, we describe three methods: Base-Pos, Post-Pos, and Base-Cut.

- **Base-Pos**: *After applying the standard* FO*, we convert all negative estimations to* 0*.*

This satisfies non-negativity, but the sum of all estimations is likely to be above 1. This reduces variance, as it turns erroneous negative estimations to 0, closer to the true value. As a result, for each individual value, Base-Pos results in an estimation that is at least as accurate as the Base method. However, this introduces systematic positive bias, because some negative noise are removed or reduced by the process, but the positive noise are never removed. This positive bias will be reflected when answering subset queries, for which Base-Pos results in biased estimations. For larger-range queries, the bias can be significant.

Lemma 1. *Base-Pos will introduce positive bias to all values.*

Proof: The outputs of standard FO are unbiased estimation, which means for any v,

$$f_v = \mathrm{E}\left[\,\tilde{f}_v\,\right] = \mathrm{E}\left[\,\tilde{f}_v \cdot 1_{\tilde{f}_v \geq 0}\,\right] + \mathrm{E}\left[\,\tilde{f}_v \cdot 1_{\tilde{f}_v < 0}\,\right]$$

where f_v is the true frequency of value v and $\tilde{f}_v$ is the estimated frequency of value v (or the result of Base). As Base-Pos changes all negative estimated frequencies to 0, we have

$$\mathrm{E}\left[\,f_v\,\right] = \mathrm{E}\left[\,\tilde{f}_v \cdot 1_{\tilde{f}_v \geq 0}\,\right]$$

After enforcing non-negativity constraints, the bias will be $\mathrm{E}\left[\,f_v\,\right] - f_v > 0$. □

- **Post-Pos**: *For each query result, if it is negative, we convert it to* 0*.*

This method does not post-process the estimated distribution. Rather, it post-processes each query result individually. For subset queries, as the results are typically positive, Post-

Pos is similar to Base. On the other hand, when the query is on a single item, Post-Pos is equivalent to Base-Pos.

Post-Pos still introduces a positive bias, but the bias would be smaller for subset queries. However, Post-Pos may give inconsistent answers in the sense that the query result on $A \cup B$, where A and B are disjoint, may not equal the addition of the query results for A and B separately.

- **Base-Cut**: *After standard* FO*, convert everything below some sensitivity threshold to 0.*

The original design goal for frequency oracles is to recover frequencies for *frequent* values, and oftentimes there is a sensitivity threshold so that only estimations above the threshold are considered. Specifically, for each value, we compare its estimation with a threshold (also presented in Equation 3.13)

$$T = \Phi^{-1}\left(1 - \frac{\alpha}{d}\right)\sqrt{\mathrm{Var}^*} \tag{8.1}$$

where d is the domain size, Phi^{-1} is the inverse of cummulative distribution function of the standard normal distribution, and Var^* is the standard deviation of the LDP mechanism (i.e., as in Equation (3.3)). By Base-Cut, estimations below the threshold are considered to be noise. When using such a threshold, for any value $v \in D$ whose original count is 0, the probability that it will have an estimated frequency above T (or the probability a zero-mean Gaussian variable with standard deviation δ is above T) is at most $\frac{\alpha}{d}$. Thus when we observe an estimated frequency above T, the probability that the true frequency of the value is 0 is (by union bound) at most $d \times \frac{\alpha}{d} = \alpha$. In [2], it is recommended to set $\alpha = 5\%$, following conventions in the statistical community.

Empirically we observe that $\alpha = 5\%$ performs poorly, because such a threshold can be too high when the population size is not very large and/or the ϵ is not large. A large threshold results in all except for a few estimations to be below the threshold and set to 0. We note that the choice of α is trading off false positives with false negatives. Given a large domain, there are likely between several and a few dozen values that have quite high frequencies,

with most of the remaining values having low true counts. We want to keep an estimation if it is a lot more likely to be from a frequent value than from a very low frequency one. In this paper, we choose to set $\alpha = 2$, which ensures that the expected number of *false positives*, i.e., values with very low true frequencies but estimated frequencies above T, to be around 2. If there are around 20 values that are truly frequent and have estimated frequencies above T, then ratio of true positives to false positives when using this threshold is 10:1.

This method ensures that all estimations are non-negative. It does not ensure that the sum of estimations is 1. The resulting estimations are either high (above the chosen threshold) or zero. The estimation for each item with non-zero frequency is subject to two bias effects. The negative bias effect is caused by the situation when the estimations are cut to zero. The positive effect is when large positive noise causes the estimation to be above the threshold, the resulting estimation is higher than true frequency.

8.1.2 Normalization Method

We now explore several methods that normalize the estimated frequencies of the whole domain to ensure that the sum of the estimates equals 1. When the estimations are normalized to sum to 1, the sum of the biases over the whole domain has to be 0.

Lemma 2. *If a normalization method adjusts the unbiased estimates so that they add up to* 1*, the sum of biases it introduces over the whole domain is* 0*.*

PROOF: Denote f_v as the estimated frequency of value v after post-processing. By linearity of expectations, we have

$$\sum_{v \in D} (\mathrm{E}[\, f_v \,] - f_v) = \mathrm{E}\left[\sum_{v \in D} f_v \right] - \sum_{v \in D} f_v = \mathrm{E}[\, 1 \,] - 1 = 0$$

□

One standard way to do such normalization is through additive normalization:

- **Norm**: *After standard* FO*, add* δ *to each estimation so that the overall sum is 1.*

The method is formally proposed for the centralized setting [17] of DP and is used in the local setting, e.g., [16], [40]. Note the method does not enforce non-negativity. For GRR, Hadamard Response, and Subset Selection, this method actually does nothing, since each user reports a single value, and the estimations already sum to 1. For OLH, however, each user reports a randomly selected subset whose size is a random variable, and Norm would change the estimations. It can be proved that Norm is unbiased:

Lemma 3. *Norm provides unbiased estimation for each value.*

PROOF: By the definition of Norm, we have $\sum_{v\in D} f_v = \sum_{v\in D}(\tilde{f}_v + \delta) = 1$. As the frequency oracle outputs unbiased estimation, i.e., $\mathrm{E}\left[\,\tilde{f}_v\,\right] = f_v$, we have

$$\begin{aligned}
\mathrm{E}\left[\sum_{v\in D} f_v\right] &= 1 = \mathrm{E}\left[\sum_{v\in D}(\tilde{f}_v + \delta)\right] \\
&= \sum_{v\in D} \mathrm{E}\left[\,\tilde{f}_v\,\right] + d\cdot \mathrm{E}\left[\,\delta\,\right] = 1 + d\cdot \mathrm{E}\left[\,\delta\,\right] \\
\Longrightarrow \mathrm{E}\left[\,\delta\,\right] &= 0
\end{aligned}$$

Thus $\mathrm{E}\left[\,f_v\,\right] = \mathrm{E}\left[\,\tilde{f}_v + \delta\,\right] = \mathrm{E}\left[\,\tilde{f}_v\,\right] + 0 = f_v$. □

Besides sum-to-one, if a method also ensures non-negativity, we first state that it introduces positive bias to values whose frequencies are close to 0.

Lemma 4. *If a normalization method adjusts the unbiased estimates so that they add up to* 1 *and are non-negative, then it introduces positive biases to values that are sufficiently close to* 0.

PROOF: As the estimates are non-negative and sum up to 1, some of the estimates must be positive. For a value close to 0, there exists some possibility that its estimation is positive; but the possibility its estimation is negative is 0. Thus the expectation of its estimation is positive, leading to a positive bias. □

Lemma 4 shows the biases for any method that ensures both constraints cannot be all zeros. Thus different methods are essentially different ways of distributing the biases. Next we present three such normalization methods.

- **Norm-Mul**: *After standard* FO*, convert negative value to 0. Then multiply each value by a multiplicative factor so that the sum is 1.*

More precisely, given estimation vector $\tilde{\mathbf{f}}$ (we use the bold font to denote the vector of values that corresponds to all possible values in the domain, i.e., $\mathbf{f} = \langle f_v \rangle_{v \in D}$), we find γ such that

$$\sum_{v \in D} \max(\gamma \times \tilde{f}_v, 0) = 1,$$

and assign $f_v = \max(\gamma \times \tilde{f}_v, 0)$ as the estimations. This results in a consistent FO. Kairouz et al. [41] evaluated this method and it performs well when the underlying dataset distribution is smooth. This method results in positive biases for low-frequency items, but negative biases for high-frequency items. Moreover, the higher an item's true frequency, the larger the magnitude of the negative bias. The intuition is that here γ is typically in the range of $[0, 1]$; and multiplying by a factor may result in the estimation of high frequency values to be significantly lower than their true values. When the distribution is skewed, which is more interesting in the LDP case, the method performs poorly.

- **Norm-Sub**: *After standard* FO*, convert negative values to 0, while maintaining overall sum of 1 by adding δ to each remaining value.*

More precisely, given estimation vector $\tilde{\mathbf{f}}$, we want to find δ such that

$$\sum_{v \in D} \max(\tilde{f}_v + \delta, 0) = 1$$

Then the estimation for each value v is $f_v = \max(\tilde{f}_v + \delta, 0)$. This extends the method Norm and results in consistency. Norm-Sub was used by Kairouz et al. [41] and Bassily [42] to process results for some FO's. Under Norm-Sub, low-frequency values have positive biases, and high-frequency items have negative biases. The distribution of biases, however, is more even when compared to Norm-Mul.

- **Norm-Cut**: *After standard* FO*, convert negative and small positive values to 0 so that the total sums up to 1.*

We note that under Norm-Sub, higher frequency items have higher negative biases. One natural idea to address this is to turn the low estimations to 0 to ensure consistency, without changing the estimations of high-frequency values. This is the idea of Norm-Cut. More precisely, given the estimation vector $\tilde{\mathbf{f}}$, there are two cases. When $\sum_{v\in D}\max(\tilde{f}_v, 0) \leq 1$, we simply change each negative estimations to 0. When $\sum_{v\in D}\max(\tilde{f}_v, 0) > 1$, we want to find the smallest θ such that

$$\sum_{v\in D|\tilde{f}_v\geq\theta} \tilde{f}_v \leq 1$$

Then the estimation for each value v is 0 if $\tilde{f}_v < \theta$ and $\tilde{f}_v$ if $\tilde{f}_v \geq \theta$. This is similar to Base-cut in that both methods change all estimated values below some thresholds to 0. The differences lie in how the threshold is chosen. This results in non-negative estimations, and typically results in estimations that sum up to 1, but might result in a sum < 1.

8.1.3 Constrained Least Squares

From a more principled point of view, we note that what we are doing here is essentially solving a Constraint Inference (CI) problem, for which CLS (Constrained Least Squares) is a natural solution. This approach was proposed in [17] but without the constraint that the estimates are non-negative (and it leads to Norm). Here we revisit this approach with the consistency constraint (i.e., requirements of both non-negativity and sum-up-to-one).

- **CLS**: *After standard* FO*, use least squares with constraints (summing-to-one and non-negativity) to recover the values.*

Specifically, given the estimates $\tilde{\mathbf{f}}$ by FO, the method outputs $\mathbf{f}$ that is a solution of the following problem:

$$\begin{aligned} \text{minimize: } & ||\mathbf{f} - \tilde{\mathbf{f}}||_2 \\ \text{subject to: } & \forall_v f_v \geq 0 \\ & \sum_v f_v = 1 \end{aligned}$$

We can use the KKT condition [30], [31] to solve the problem. We first augment the optimization target with the following equations:

$$\begin{aligned} \text{minimize} \quad & \sum_v (f_v - \tilde{f}_v)^2 + a + b \\ \text{where} \quad & \sum_v f_v = 1, \ \ \forall v : 0 \leq f_v \leq 1, \\ & a = \mu \cdot \sum_v f_v, b = \sum_v \lambda_v \cdot f_v, \forall v : \lambda_v \cdot f_v = 0. \end{aligned}$$

Since $b = 0$, and $a = \mu$ is a constant, the condition that minimizing the target is unchanged. Given that the target is convex, we can find the minimum by taking the partial derivative with respect to each variable:

$$\begin{aligned} & \frac{\partial \left[\sum_v (f_v - \tilde{f}_v)^2 + a + b\right]}{\partial f_v} = 0 \\ \Longrightarrow & \ 2(f_v - \tilde{f}_v) + \mu + \lambda_v = 0 \\ \Longrightarrow & \ f_v = \tilde{f}_v - \frac{1}{2}(\mu + \lambda_v) \end{aligned}$$

Now suppose there is a subset of domain $D_0 \subseteq D$ s.t., $\forall v \in D_0, f_v = 0$ and $\forall v \in D_1 = D \setminus D_0, f_v > 0 \wedge \lambda_v = 0$. By summing up f_v for all $v \in D_1$, we have

$$1 = \sum_{v \in D_1} \tilde{f}_v - \frac{|D_1|\mu}{2}$$

Thus for all $v \in D_1$, we can use the formula

$$f_v = \tilde{f}_v - \frac{1}{|D_1|}\left(\sum_{v \in D_1} \tilde{f}_v - 1\right)$$

to derive the estimate f_v for value $v \in D_1$, and $f_v = 0$ for $v \in D_0$. One can also find D_0 using a similar approach when dealing with MLE. And it can also be verified $\sum_v f_v = 1$.

Given the derivation above, we can see that Norm-Sub is the solution to the Constraint Least Square (CLS) formulation to the problem, and $\delta = -\frac{1}{|D_1|}\left(\sum_{v \in D_1} \tilde{f}_v - 1\right)$ is the δ we want to find in Norm-Sub.

8.1.4 Maximum Likelihood Estimation

Another more principled way of looking into this problem is to view it as recovering distributions given some LDP reports. For this problem, one standard solution is Bayesian inference. In particular, we want to find the $\mathbf{f}$ such that

$$\mathsf{Pr}\left[\mathbf{f}|\tilde{\mathbf{f}}\right] = \frac{\mathsf{Pr}\left[\tilde{\mathbf{f}}|\mathbf{f}\right] \cdot \mathsf{Pr}\left[\mathbf{f}\right]}{\mathsf{Pr}\left[\tilde{\mathbf{f}}\right]} \tag{8.2}$$

is maximized. Note that we require $\mathbf{f}$ satisfies $\forall_v f_v \geq 0$ and $\sum_v f_v = 1$. In Equation (8.2), $\mathsf{Pr}\left[\mathbf{f}\right]$ is the prior, and the prior distribution influence the result. In our setting, as we assume there is no such prior, $\mathsf{Pr}\left[\mathbf{f}\right]$ is uniform. That is, $\mathsf{Pr}\left[\mathbf{f}\right]$ is a constant. The denominator $\mathsf{Pr}\left[\tilde{\mathbf{f}}\right]$ is also a constant that does not influence the result. As a result, we are seeking for $\mathbf{f}$ which is the maximal likelihood estimator (MLE), i.e., $\mathsf{Pr}\left[\tilde{\mathbf{f}}|\mathbf{f}\right]$ is maximized.

For this method, Peter et al. [41] derived the exact MLE solution for GRR and RAPPOR [2]. We compute $\mathsf{Pr}\left[\tilde{\mathbf{f}}|\mathbf{f}\right]$ using the approximation of $\tilde{\mathbf{f}}$, which is a set of independent

random variables, and each component $\tilde{f}_v$ follows Gaussian distribution with mean f_v and variance $\sigma_v'^2$. The likelihood of $\tilde{\mathbf{f}}$ given $\mathbf{f}$ is thus

$$\begin{aligned}&\mathsf{Pr}\left[\tilde{\mathbf{f}}|\mathbf{f}\right] = \prod_v \mathsf{Pr}\left[\tilde{f}_v|f_v\right] \\ \approx &\prod_v \frac{1}{\sqrt{2\pi\sigma_v'^2}} \cdot e^{-\frac{(f_v-\tilde{f}_v)^2}{2\sigma_v'^2}} = \frac{1}{\sqrt{2\pi\prod_v \sigma_v'^2}} \cdot e^{-\sum_v \frac{(f_v-\tilde{f}_v)^2}{2\sigma_v'^2}}. \end{aligned} \tag{8.3}$$

To differentiate from [41], we call it MLE-Apx.

- **MLE-Apx**: *First use standard* FO*, then compute the MLE with constraints (summing-to-one and non-negativity) to recover the values.*

From Equation (8.3), we first simplify the exponent plugging in the value of σ_v as in Equation (3.2):

$$\sum_v \frac{(f_v-\tilde{f}_v)^2}{2\sigma_v'^2} = \frac{n}{2}\sum_v \frac{(f_v-\tilde{f}_v)^2(p-q)^2}{q(1-q)+f_v(p-q)(1-p-q)}$$

The factor $\frac{n}{2}$ in the exponent ensures that for large n the exponent will vary the most with $\mathbf{f}$, which dominates the coefficient $\frac{1}{\sqrt{2\pi\prod_v \sigma_v'^2}}$. Thus approximately we find $\mathbf{f}$ that achieves the following optimization goal:

$$\begin{aligned}\text{minimize: } &\sum_v \frac{(f_v-\tilde{f}_v)^2(p-q)^2}{q(1-q)+f_v(p-q)(1-p-q)} \\ \text{subject to: } &\sum_v f_v = 1, \\ &\forall v, 0 \le f_v \le 1.\end{aligned}$$

Now to solve this optimization problem, we can also use the KKT condition [30], [31], We first augment the optimization target with the following equations:

$$\begin{aligned}\text{minimize} \quad &\sum_v \frac{(f_v-\tilde{f}_v)^2(p-q)^2}{q(1-q)+f_v(p-q)(1-p-q)} + a + b \\ \text{where} \quad &\sum_v f_v = 1, \ \ \forall v : 0 \le f_v \le 1, \\ &a = \mu \cdot \sum_v f_v, b = \sum_v \lambda_v \cdot f_v, \forall v : \lambda_v \cdot f_v = 0.\end{aligned}$$

Since $b = 0$, and $a = \mu$ is a constant, the condition for minimizing the target is unchanged. Given that the target is convex, we can find the minimum by taking the partial derivative with respect to each variable:

$$\begin{aligned}
&\frac{\partial \left[\sum_v \frac{(f_v - \tilde{f}_v)^2 (p-q)^2}{q(1-q) + f_v(p-q)(1-p-q)} + a + b\right]}{\partial f_v} \\
=&\frac{-(f_v - \tilde{f}_v)^2 (p-q)^2 \cdot (p-q)(1-p-q)}{(q(1-q) + f_v(p-q)(1-p-q))^2} \\
+&\frac{2(f_v - \tilde{f}_v)(p-q)^2}{q(1-q) + f_v(p-q)(1-p-q)} + \mu + \lambda_v = 0
\end{aligned}$$

Define a temporary notation

$$\begin{aligned}
x_v =& \frac{(f_v - \tilde{f}_v)(p-q)}{q(1-q) + f_v(p-q)(1-p-q)} \\
\text{so that } f_v =& \frac{q(1-q)x_v + \tilde{f}_v(p-q)}{p - q - (p-q)(1-p-q)x_v}
\end{aligned} \tag{8.4}$$

With x_v, we can simplify the previous equation:

$$(p-q)(1-p-q)x_v^2 - 2(p-q)x_v - \mu - \lambda_v = 0 \tag{8.5}$$

Now suppose there is a subset of domain $D_0 \subseteq D$ s.t., $\forall v \in D_0, f_v = 0$ and $\forall v \in D_1 = D \setminus D_0, f_v > 0$ and $\lambda_v = 0$. Thus for those $v \in D_1$, solution of x_v in Equation (8.5) does not depend on v. We solve x_v by summing up f_v for all $v \in D_1$:

$$\begin{aligned}
\sum_{v \in D_1} f_v =& 1 = \sum_{v \in D_1} \frac{q(1-q)x_v + \tilde{f}_v(p-q)}{p - q - (p-q)(1-p-q)x_v} \\
=& \frac{|D_1| q(1-q)x_v + \sum_{v \in D_1} \tilde{f}_v(p-q)}{p - q + (p-q)(1-p-q)x_v} \\
\implies x_v =& \frac{\sum_{x \in D_1} \tilde{f}_v(p-q) - (p-q)}{(p-q)(1-p-q) - |D_1| q(1-q)}
\end{aligned}$$

Given x_v, we can compute f_v from Equation (8.4) for each value $v \in D_1$ efficiently; and $f_v = 0$ for $v \in D_0$. It can be verified $\sum_v f_v = 1$.

Finally, to find D_0, one initiates $D_0 = \emptyset$ and $D_1 = D$, and iteratively tests whether all values in D_1 are positive. In each iteration, for any negative a_x, x is moved from D_1 to D_0. The process terminates when no negative a_x is found for all $x \in D_1$.

In particular, we partition the domain D into D_0 and D_1, where $D_0 \cap D_1 = \emptyset$ and $D_0 \cup D_1 = D$. For $v \in D_0$, $f_v = 0$; for $v \in D_1$,

$$f_v = \frac{q(1-q)x_v + \tilde{f}_v(p-q)}{p-q-(p-q)(1-p-q)x_v} \tag{8.6}$$

where

$$x_v = \frac{\sum_{x \in D_1} \tilde{f}_v(p-q) - (p-q)}{(p-q)(1-p-q) - |D_1|q(1-q)}$$

We can rewrite Equation (8.6) as

$$f_v = \tilde{f}_v \cdot \gamma + \delta,$$

where

$$\gamma = \frac{p-q}{p-q+(p-q)(1-p-q)x_v}$$
$$\delta = \frac{q(1-q)x_v}{p-q+(p-q)(1-p-q)x_v}$$

Hence MLE-Apx appears to represent some hybrid of Norm-Sub and Norm-Mul. In evaluation, we observe that Norm-Sub and MLE-Apx give very close results, as $\gamma \sim 1$. Furthermore, when the f_v component in variance is dominated by the other component (as in Equation (3.3)), the CLS formulation is equivalent to our MLE formulation.

8.1.5 Least Expected Square Error

Jia et al. [43] proposed a method in which one first assumes that the data follows some type of distribution (but the parameters are unknown), then uses the estimates to fit the

parameters of the distribution, and finally updates the estimates that achieve expected least square.

- **Power**: *Fit a distribution, and then minimize the expected squared error.*

Formally, for each value v, the estimate $\tilde{f}_v$ given by FO is regarded as the addition of two parts: the true frequency f_v and noise following the normal distribution. The method then finds f_v that minimizes $\mathrm{E}\left[(f_v - f_v)^2 | \tilde{f}_v\right]$. To solve this problem, the authors estimate the true distribution f_v from the estimates $\tilde{\mathbf{f}}$ (where $\tilde{\mathbf{f}}$ is the vector of the $\tilde{f}_v$'s).

In particular, it is assume in [43] that the distribution follows Power-Law or Gaussian. The distributions can be determined by one or two parameters, which can be fitted from the estimation $\tilde{\mathbf{f}}$. Given $\mathsf{Pr}[x]$ as the probability $f_v = x$ from the fitted distribution, and $\mathsf{Pr}[x \sim \mathcal{N}(0, \sigma)]$ as the probability density function of x drawn from the Normal distribution with 0 mean and standard deviation σ, one can then minimize the objective. Specifically, for each value $v \in D$, output

$$f_v = \int_0^1 \frac{\mathsf{Pr}\left[(\tilde{f}_v - x) \sim \mathcal{N}(0, \sigma)\right] \cdot \mathsf{Pr}[x] \cdot x}{\int_0^1 \mathsf{Pr}\left[(\tilde{f}_v - y) \sim \mathcal{N}(0, \sigma)\right] \cdot \mathsf{Pr}[y]\, dy} dx. \tag{8.7}$$

We fit $\mathsf{Pr}[x]$ with the Power-Law distribution and call the method Power. Using this method requires knowledge and/or assumption of the distribution to be estimated. If there are too much noise, or the underlying distribution is different, forcing the observations to fit a distribution could lead to poor accuracy. Moreover, this method does not ensure the frequencies sum up to 1, as Equation (8.7) only considers the frequency of each value v independently. To make the result consistent, we use Norm-Sub to post-process results of Power, since Power is close to CLS, and Norm-Sub is the solution to CLS. We call it PowerNS.

- **PowerNS**: *First use standard* FO*, then use Power to recover the values, finally use Norm-Sub to further process the results.*

8.1.6 Summary of Methods

In summary, Norm-Sub is the solution to the Constraint Least Square (CLS) formulation to the problem. Furthermore, when the f_v component in variance is dominated by the other component (as in Equation (3.3)), the CLS formulation is equivalent to our MLE formulation. In that case, Norm-Sub is equivalent to MLE-Apx.

Table 8.1. Summary of Methods.

Method	Description	Non-neg	Sum to 1	Complexity
Base-Pos	Convert negative est. to 0	Yes	No	$O(d)$
Post-Pos	Convert negative query result to 0	Yes	No	N/A
Base-Cut	Convert est. below threshold T to 0	Yes	No	$O(d)$
Norm	Add δ to est.	No	Yes	$O(d)$
Norm-Mul	Convert negative est. to 0, then multiply γ to positive est.	Yes	Yes	$O(d)$
Norm-Cut	Convert negative and small positive est. below θ to 0.	Yes	Almost	$O(d)$
Norm-Sub	Convert negative est. to 0 while adding δ to positive est.	Yes	Yes	$O(d)$
MLE-Apx	Convert negative est. to 0, then add δ to positive est.	Yes	Yes	$O(d)$
Power	Fit Power-Law dist., then minimize expected squared error	Yes	No	$O(\sqrt{n} \cdot d)$
PowerNS	Apply Norm-Sub after Power	Yes	Yes	$O(\sqrt{n} \cdot d)$

Table 8.1 gives a summary of the methods. First of all, all of the methods preserve the frequency order of the value, i.e., $f_{v_1} \leq f_{v_2}$ iff $\tilde{f}_{v_1} \leq \tilde{f}_{v_2}$. The methods can be classifies into three classes: First, enforcing non-negativity only. Base-Pos, Post-Pos, Base-Cut, and Power fall in this category. Second, enforcing summing-to-one only. Only Norm is in this class. Third, enforcing the two requirement simultaneously. Norm-Mul, Norm-Cut, Norm-Sub, and PowerNS satisfy both requirements.

8.2 Evaluation

As we are optimizing multiple utility metrics together, it is hard to theoretically compare different methods. In this section, we run experiments to empirically evaluate these methods.

At the high level, our evaluations show that different methods perform differently in different settings, and to achieve the best utility, it may or may not be necessary to exploit all the consistency constraints. As a result, we conclude that for full-domain query, Base-Cut performs the best; for set-value query, PowerNS performs the best; and for high-frequency-value query, Norm performs the best.

8.2.1 Experimental Setup

Datasets. We run experiments on two datasets (one synthetic and one real).

- Synthetic Zipf's distribution with 1024 values and 1 million reports. We use $s = 1.5$ in this distribution.
- Emoji: The daily emoji usage data. We use the average emoji usage of an emoji keyboard [1], which gives the total count of $n = 884427$ with $d = 1573$ different emojis.

Setup. The FO protocols and post-processing algorithms are implemented in Python 3.6.6 using Numpy 1.15; and all the experiments are conducted on a PC with Intel Core i7-4790 3.60GHz and 16GB memory. Although the post-processing methods can be applied to any FO protocol, we focus on simulating OLH as it provides near-optimal utility with reasonable communication bandwidth.

Metrics. We evaluate three scenarios 1) estimate the frequency of every value in the domain (full-domain), 2) estimate the aggregate frequencies of a subset of values (set-value), and 3) estimate the frequencies of the most frequent values (frequent-value).

We use the metrics of *Mean of Squared Error* (MSE). MSE measures the mean of squared difference between the estimate and the ground truth for each (set of) value. For full-domain, we compute

$$\text{MSE} = \frac{1}{d}\sum_{v \in D}(f_v - f_v)^2.$$

For frequent-value, we consider the top k values with highest f_v instead of the whole domain D; and for set-value, instead of measuring errors for singletons, we measure errors for sets, that is, we first sum the frequencies for a set of values, and then measure the difference.

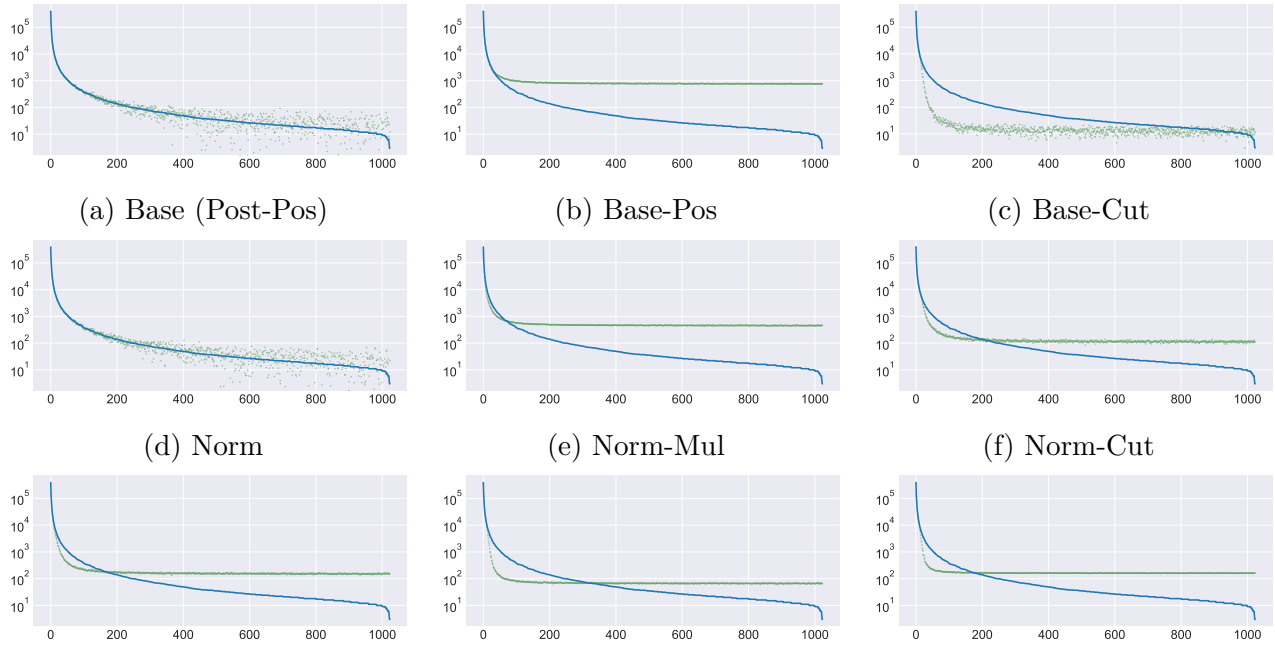
Plotting Convention. Unless otherwise specified, for each dataset and each method, we repeat the experiment 30 times, with result mean and standard deviation reported. The standard deviation is typically very small, and barely noticeable in the figures.

[1] ↑http://www.emojistats.org/, accessed 12/15/2019 10pm ET

Because there are 11 algorithms (10 post-processing methods plus Base), and for any single metric there are often multiple methods that perform very similarly, resulting their lines overlapping. To make Figures 4–8 readable, we plot results on two separate figures on the same row. On the left, we plot 6 methods, Base, Base-Pos, Post-Pos, Norm, Norm-Mul, and Norm-Sub. On the right, we plot Norm-Sub with the remaining 5 methods, MLE-Apx, Base-Cut, Norm-Cut, Power and PowerNS. We mainly want to compare the methods in the right column.

8.2.2 Bias-variance Evaluation

Figure 8.1 shows the true distribution of the synthetic Zipf's dataset and the mean of the estimations. As we plot the count estimations (instead of frequency estimations), the variance is larger (a $n^2 = 10^{12}$ multiplicative factor than the frequency estimations). We thus estimate 5000 times in order to make the mean stabilize. In Figure 8.2, we subtract the estimation mean by the ground truth and plot the difference, which representing the empirical bias. It can be seen that Base and Norm are unbiased. Base-Pos introduces systematic positive bias. Base-Cut gives unbiased estimations for the first few most frequent values, as their true frequencies are much greater than the threshold T used to cut off estimation below it to 0. As the noise is close to normal distribution, the possibility that a high-frequency value is estimated to be below T is exponentially small. The similar analysis also holds for the low-frequency values, whose estimates are unlikely to be above T. On the other hand, for values in between, the two biases compete with each other. At some point, the two effects cancel out with each other, leading to unbiased estimations. But this point is dependent on the whole distribution, and thus is hard to be found analytically. For Norm-Cut, the similar reasoning also applies, with the difference that the threshold in Norm-Cut is typically smaller. For Norm-Sub, each value is influenced by two factors: subtraction by a same amount; and converting to 0 if negative. For the high-frequency values, we mostly see the first factor; for the low-frequency values, they are mostly affected by the second factor; and for the values in between, the two factors compete against each other. We see an increasing line for Norm-Sub. Finally, Power changes little to the top estimations; but



(a) Base (Post-Pos) (b) Base-Pos (c) Base-Cut

(d) Norm (e) Norm-Mul (f) Norm-Cut

(g) Norm-Sub (h) Power (i) PowerNS

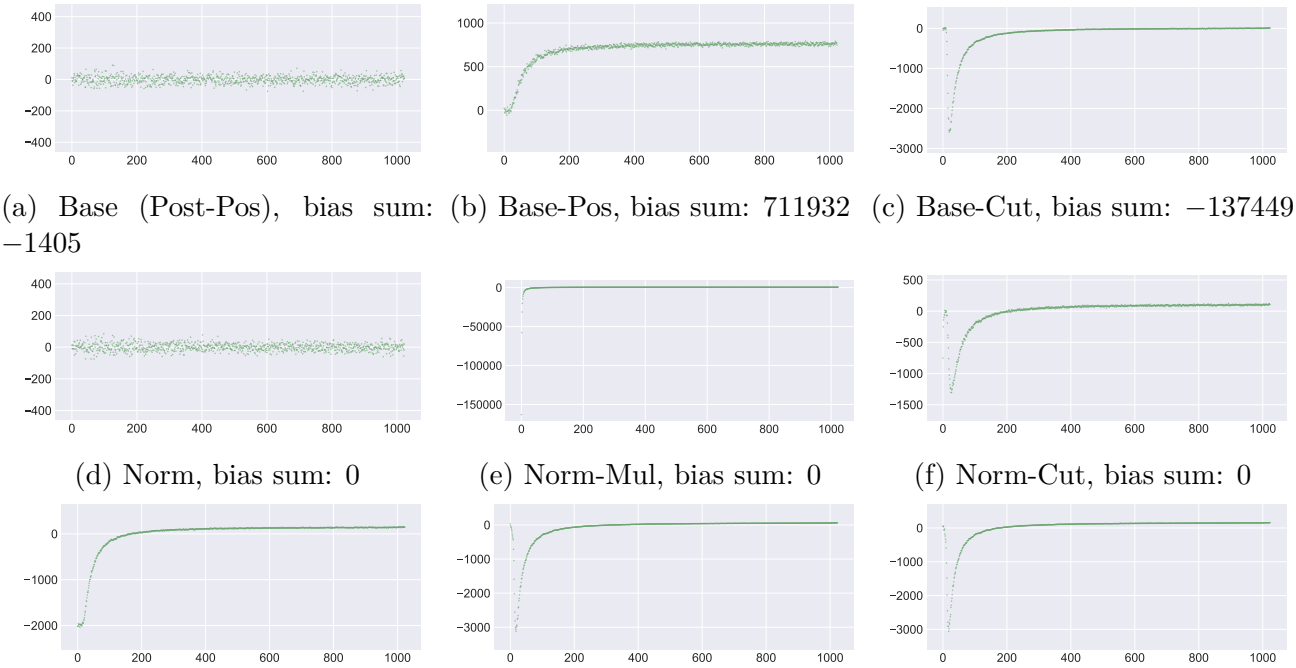
Figure 8.1. Log-scale distribution of the Zipf's dataset fixing $\epsilon = 1$, the x-axes indicates the sorted value index and the y-axes is its count. The blue line is the ground truth; the green dots are estimations by different methods.

more to the low ones, thus leading to a similar shape as Norm-Cut. The shape of PowerNS is close to Power because PowerNS applies Norm-Sub, which subtract some amount to the estimations, after Power.

Figure 8.3 shows the variance of the estimations among the 5000 runs. First of all, the variance is similar for all the values in Base and Norm, with Norm being slightly better (smaller) than Base. For all other methods, the variance drops with the rank, because for low-frequency values, their estimates are mostly zeros.

8.2.3 Full-domain Evaluation

Figure 8.4 shows MSE when querying the frequency of every value in the domain. Note that The MSE is composed of the (square of) bias shown in Figure 8.2 and variance in Figure 8.3. We vary ϵ from 0.2 to 4. Let us fist focus on the figures on the left. Base performs very close to Norm, since the adjustment of Norm can be either positive or negative as the



(a) Base (Post-Pos), bias sum: −1405 (b) Base-Pos, bias sum: 711932 (c) Base-Cut, bias sum: −137449

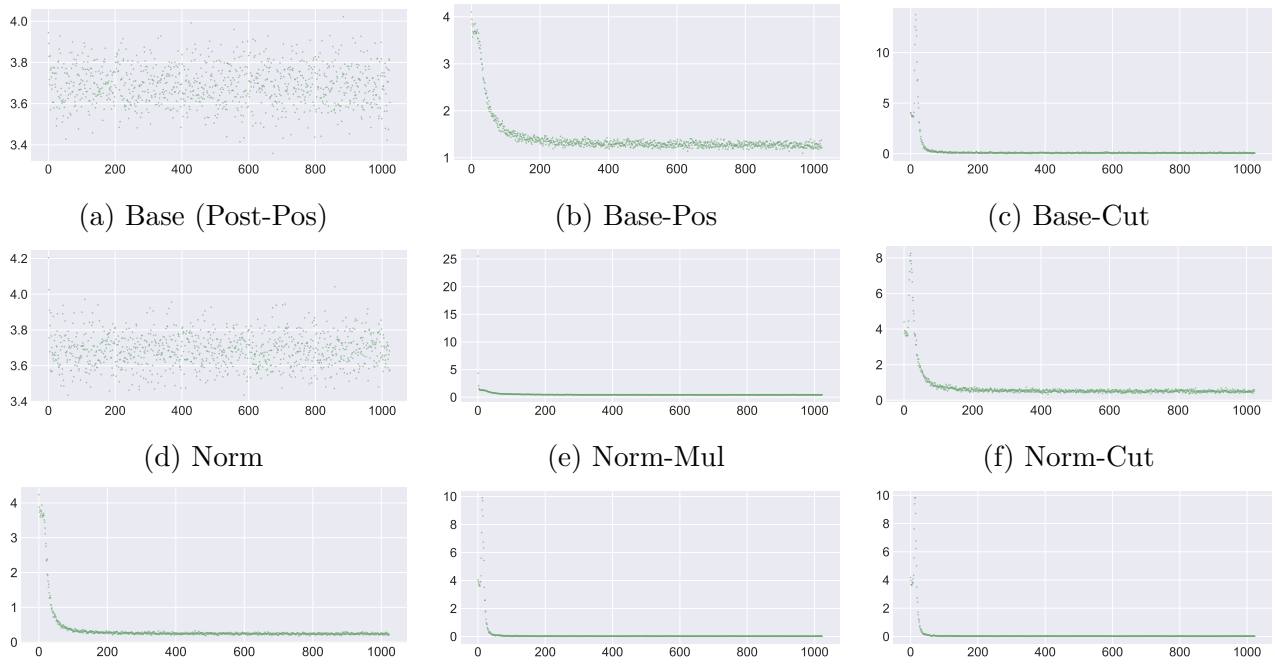
(d) Norm, bias sum: 0 (e) Norm-Mul, bias sum: 0 (f) Norm-Cut, bias sum: 0

(g) Norm-Sub, bias sum: 0 (h) Power, bias sum: −96332 (i) PowerNS, bias sum: 0

Figure 8.2. Bias of count estimation for the Zipf's dataset fixing $\epsilon = 1$.

expected value of the estimation sum is 1. As Base-Pos (which is equivalent to Post-Pos in this setting) converts negative results to 0, its MSE is around half that of Base (note the y-axis is in log-scale). Norm-Sub is able to reduce the MSE of Base by about a factor of 10 and 100 in the Zipfs and Emoji dataset respectively. Norm-Mul behaves differently from other methods. In particular, the MSE decreases much slower than other methods. This is because Norm-Mul multiplies the original estimations by the same factor. The higher the estimate, the greater the adjustment. Since the estimations are individually unbiased, this is not the correct adjustment.

For the right part of Figure 8.4, we observe that, Norm-Sub and MLE-Apx perform almost exactly the same, validating the prediction from theoretical analysis. Norm-Sub, MLE-Apx, Power, PowerNS, and Base-Cut perform very similarly. In these two datasets, PowerNS performs the best. Note that PowerNS works well when the distribution is close to Power-Law. For an unknown distribution, we still recommend Base-Cut. This is because if one considers average accuracy of all estimations, the dominating source of errors comes from the fact many values have true frequencies close or equal to 0 are randomly perturbed.



(a) Base (Post-Pos) (b) Base-Pos (c) Base-Cut

(d) Norm (e) Norm-Mul (f) Norm-Cut

(g) Norm-Sub (h) Power (i) PowerNS

Figure 8.3. Variance of count estimation of the Zipf's dataset fixing $\epsilon = 1$. The y-axes are scaled down by $n = 10^6$ (a value a in the figure represents $a \cdot 10^6$).

And Base-Cut maintains the high-frequency values unchanged, and converts results below a threshold T to 0. Norm-Cut also converts low estimations to 0, but the threshold θ is likely to be lower than T, because θ is chosen to achieve a sum of 1.

Benefit of Post-Processing. We demonstrate the benefit of post-processing by measuring the relationship between n and n, so that n records with post-processing can achieve the same accuracy for n records without it. In particular, we vary n and measure the errors for different methods. We then calculate n using Equation 3.2. In particular, the analytical MSE for n records is

$$\begin{aligned}\frac{1}{d}\sum_v \sigma_v^2 =& \frac{q(1-q)}{n(p-q)^2} + \frac{1}{d}\sum_v \frac{f_v(1-p-q)}{n(p-q)} \\ =& \frac{q(1-q)}{n(p-q)^2} + \frac{1}{d}\frac{1-p-q}{n(p-q)}.\end{aligned}$$

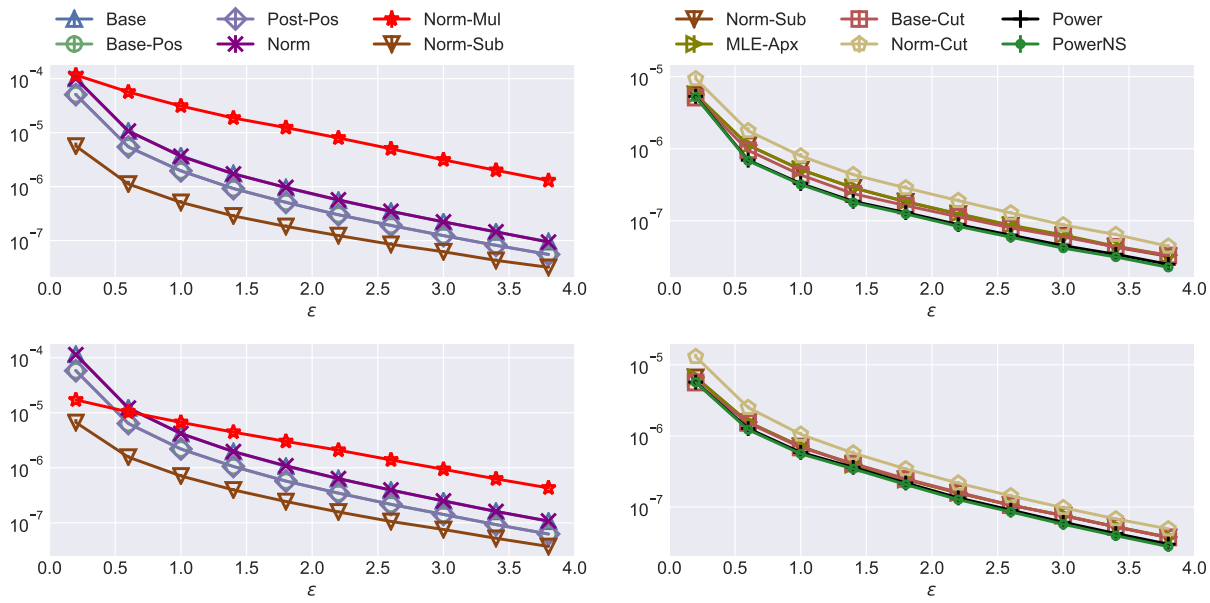


Figure 8.4. MSE results on full-domain estimation, varying ϵ from 0.2 to 4. The top row is for Zipf's distribution and the bottom row is for the Emoji dataset.

Given the empirical MSE, we can obtain n that achieves the same error analytically. Note that the MSE does not depend on the distribution. Thus we only evaluate on the Zipf's dataset. The result is shown in Figure 8.5. We vary the size of the dataset n and plot the value of n (note that the x-axes are in the scale of 10^6 and y-axes are 10^7). The higher the line, the better the method performs. Base and Norm are two straight lines with the slope of 1, verifying the analytical variance. The y value for Norm-Mul grows even slower than Base, indicating the harm of using Norm-Mul as a post-processing method. The performance of the other methods follow the similar trend of the full-domain MSE (as shown in the upper row of Figure 8.4), with PowerNS gives the best performance, which saves around 90% of users.

8.2.4 Set-value Evaluation

Estimating set-values plays an important role in the interactive data analysis setting (e.g., estimating which category of emoji's is more popular). Keeping $\epsilon = 1$, we evaluate the performance of different methods by changing the size of the set. For the set-value queries,

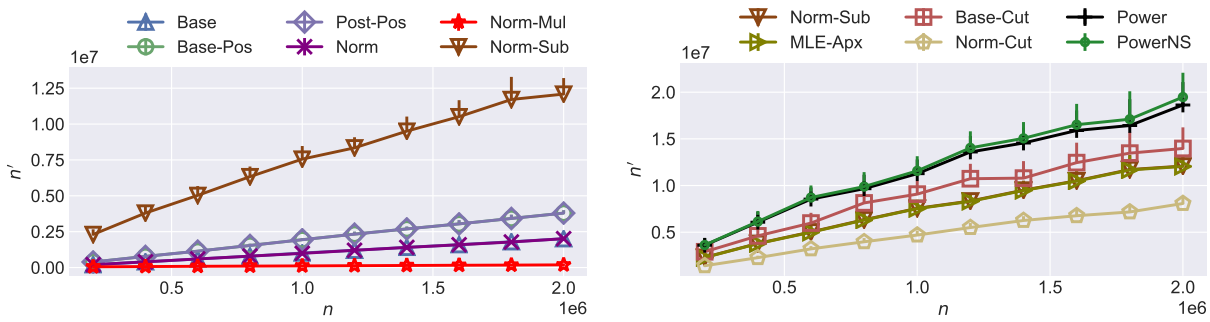


Figure 8.5. MSE results on full-domain estimation on Zipfs dataset, comparing n with n, fixing $\epsilon = 1$ while varying n from 0.2×10^6 to 2.0×10^6. Three pairs of methods have similar performance: Base and Norm, Base-Pos and Post-Pos, Norm-Sub and MLE-Apx.

we uniformly sample $\rho\% \times |D|$ elements from the domain and evaluate the MSE between the sum of their true frequencies and estimated frequencies. Formally, define $D_{s\rho}$ as the random subset of D that has $\rho\% \times |D|$ elements; and define $f_{D_{s\rho}} = \sum_{v \in D_{s\rho}} f_v$. We sample $D_{s\rho}$ multiple times and measure MSE between $f_{D_{s\rho}}$ and $f_{D_{s\rho}}$. Overall, the error MSE of set-value queries is greater than that for the full-domain evaluation, because the error for individual estimation accumulates.

Vary ρ from 10 to 90. Following the layout convention, we show results for set-value estimations in Figure 8.6, where we first vary ρ from 10 to 90. Overall, the approaches that exploits the summing-to-1 requirement, including Norm, Norm-Mul, Norm-Sub, MLE-Apx, Norm-Cut, and PowerNS, perform well, especially when ρ is large. Moreover, their MSE is symmetric with $\rho = 50$. This is because as the results are normalized, estimating set-values for $\rho > 50$ equals estimating the rest. When $\rho = 90$, the best norm-based method, PowerNS, outperforms any of the non-norm based methods by at least 2 orders of magnitude.

For each specific method, it is observed the MSE for Base-Pos is higher than other methods, because it only turns negative estimates to 0, introducing systematic bias. Post-Pos is slightly better than Base, as it turns negative query results to 0. In the settings we evaluated, Base-Cut also outperforms Base; this happens because converting estimates below the threshold T to 0 is more likely to make the summation f_D close to one. Finally, Power

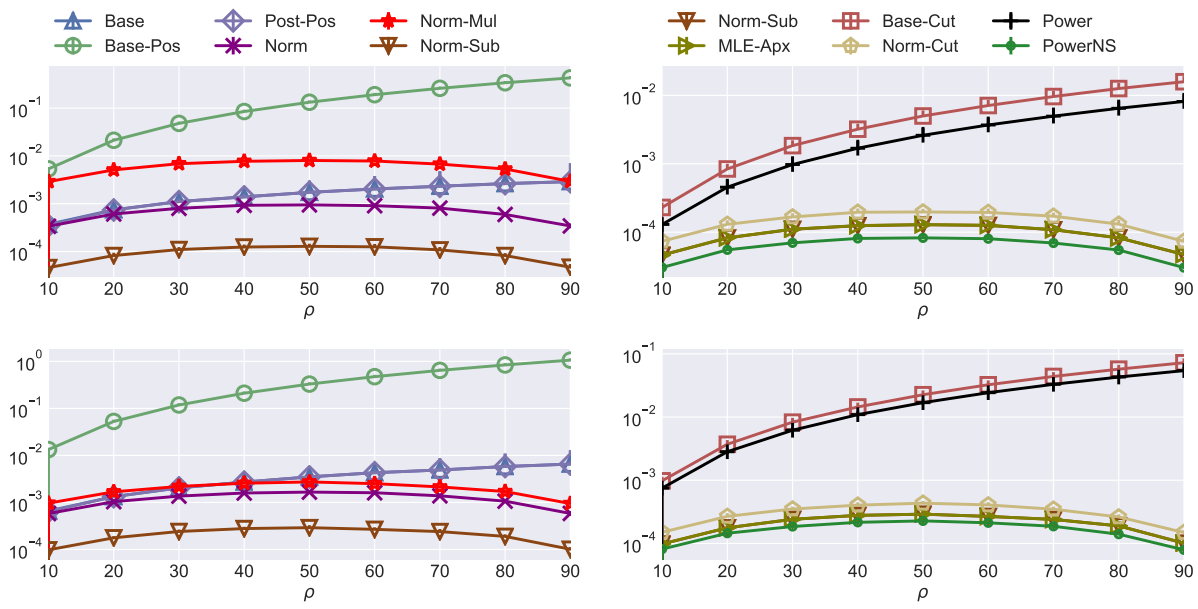


Figure 8.6. MSE results on set-value estimation, varying set size percentage ρ from 10 to 90, fixing $\epsilon = 1$. Top row is Zipf's and bottom row is Emoji.

only converts negative estimations to be positive, introducing systematic bias; PowerNS further makes them sum to 1, thus achieving better utility than all other methods.

Vary ρ from 1 to 10. Having examined the performance of set-queries for larger ρ, we then vary ρ from 1 to 10 and demonstrate the results in Figure 8.7. Within this ρ range, the errors of all methods increase with ρ, which is as expected. When ρ becomes small, the performance of different methods approaches to that of full-domain estimation.

Norm-Cut varies the threshold so that after cutting, the remaining estimates sum up to one. Thus the performance of Norm-Cut is better than Base-Cut especially when $\rho \geq 2$. Intuitively, the norm-based methods should perform better answering set-queries. But Norm-Mul does not. This is because the multiplication operation reduces the large estimates a lot, making them biased. This also demonstrates that enforcing sum-to-one is not enough. Different approaches perform significantly different.

Fixed set queries. Besides random set queries, we include a case study of fixed subset queries for the Emoji dataset. The queries ask the frequency of each category[2]. There are 68

[2]↑https://data.world/kgarrett/emojis

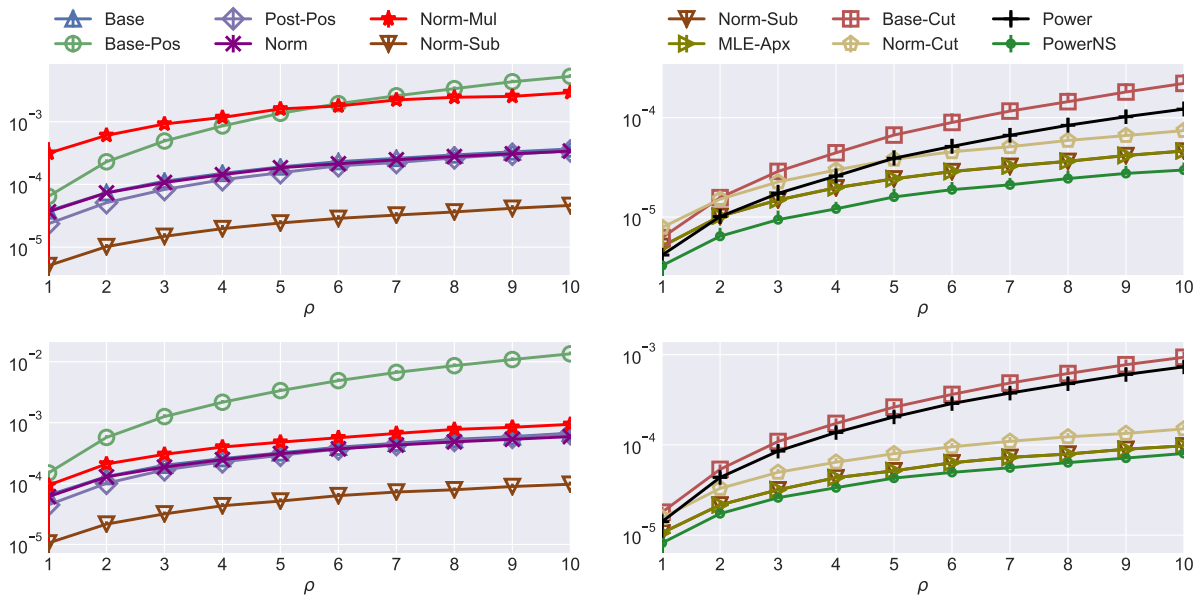


Figure 8.7. MSE results on set-value estimation, varying set size percentage ρ from 1 to 10, fixing $\epsilon = 1$. Top row is Zipf's and bottom row is Emoji.

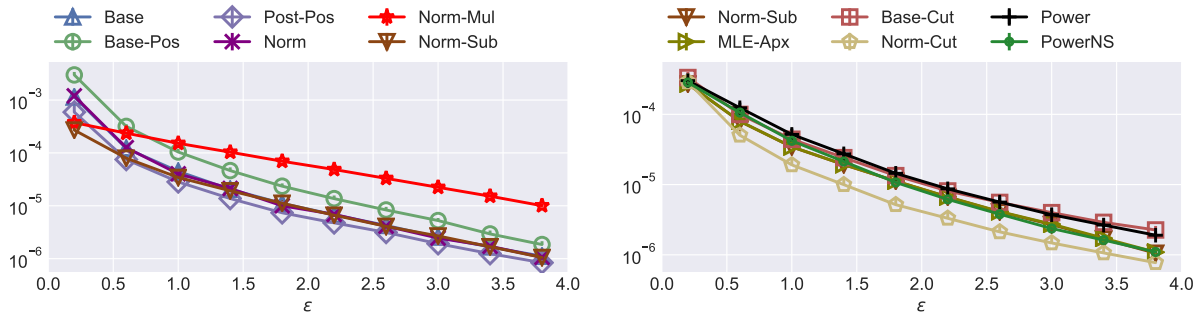


Figure 8.8. MSE results on set-case estimation for the Emoji dataset, varying ϵ from 0.2 to 4.

categories with the mean of 10.4 items per set. The MSE varying ϵ is reported in Figure 8.8. It is interesting to see that the Post-Pos works best in the left sub-figure, and Norm-Cut from the right performs even better, especially when $\epsilon < 3$. This indicates the set-queries contain values that are infrequent.

Choosing the method on synthetic dataset. As the optimal method in fixed set-values (as shown in Figure 8.8) is different from random set-values (shown in Figure 8.6 and 8.7), we investigate whether we can select the optimal post-processing method given the query and the LDP reports. In particular, we first fit a synthetic dataset from the estimation,

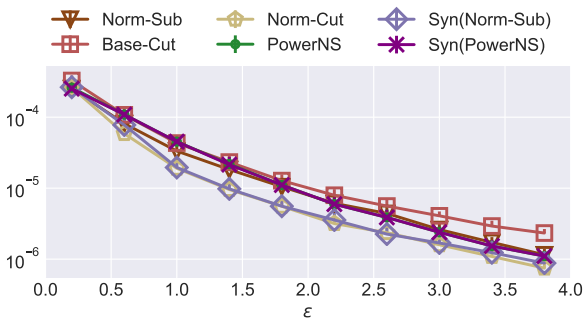


Figure 8.9. Synthetic estimation for set-case query on the Emoji dataset.

then we simulate the data collection and estimation process multiple times, with different post-processing methods, and we calculate the errors taking the synthesized dataset as the ground truth. Figure 8.9 shows the result. Note that as we generate the synthetic dataset from the estimated distribution, the distribution itself should be consistent (non-negative and sum up to 1). We select Norm-Sub and PowerNS to process the estimated distribution first. These two methods perform well on full-domain and random set-value queries.

From the figure we can see that if the results are processed by Norm-Sub, the optimal method can be find quite accurately; if PowerNS is used, PowerNS will be selected. The reason is that PowerNS makes the distribution more close to the prior of Power-Law distribution, while Norm-Sub does not.

8.2.5 Frequent-value Evaluation

Finally, we evaluate different methods varying the top values to be considered. Define D_{tk} as $\{v \in D \mid f_v \text{ ranks top } k\}$. We measure MSE between $(f_v)_{v \in D_{tk}}$ and $(f_v)_{v \in D_{tk}}$ for different values of k (from 2 to 32), fixing $\epsilon = 1$. Note that neither the frequency oracle nor the subsequent post-processing operation is aware of D_{tk}.

From the left column of Figure 8.10, we observe that Base, Base-Pos, Post-Pos, and Norm perform consistently well for different k, as the first three methods do nothing to the top values, and Norm touches them in an unbiased way. Norm-Mul performs at least 10× worse than any other methods because it reduces the higher estimations a lot. Norm-Sub performs

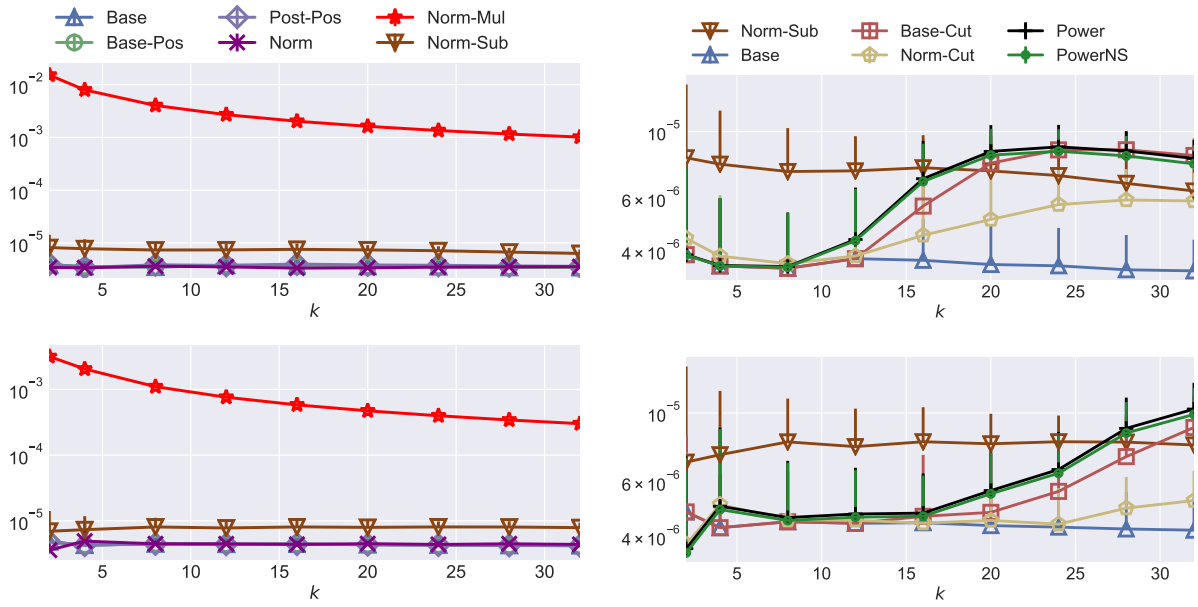


Figure 8.10. MSE results on top-k value estimation varying k from 2 to 32, fixing $\epsilon = 1$. Top row is Zipf's and bottom row is Emoji.

worse than Base, but better than Norm-Mul, because the same amount is subtracted from every estimate, regardless of k.

To give a better comparison, we plot both Base and Norm-Sub to the right (i.e., we ignore MLE-Apx for now, as it performs the same as Norm-Sub). These two methods have consistent MSE for different k. The rest four methods, Base-Cut, Norm-Cut, Power, and PowerNS, all have MSE that grows with k. In particular, for Base-Cut, a fixed threshold T (in Equation (3.13)) is used and estimates below it is converted to 0. This also suggests that at $\epsilon = 1$, around 10 values can be reliably estimated. This also happens to Norm-Cut for the similar reason. As Norm-Cut is better than Base-Cut, it suggests the threshold used in Norm-Cut is smaller than that in Base-Cut. If T is reduced, MSE of Base-Cut can be lowered until it matches that of Norm-Cut. Thus T is actually a tradeoff between frequent values and set-values. In practice, if the desired k is known in advance, one can set T to be the k-th highest estimated value. Finally, the performances of Power and PowerNS are similar, and they are worse than Base-Cut, especially when $k > 10$.

8.2.6 Discussion

In summary, we evaluate the 10 post-processing methods on different datasets, for different tasks, and varying different parameters. We now summarize the findings and present guidelines for using the post-processing methods.

With the experiments, we verify the connections among the methods: Norm-Sub and MLE-Apx perform similarly, and Base and Norm performs similarly.

The best choice for post-processing method depends on the queries one wants to answer. If set-value estimation is needed, one should use PowerNS. When the set is fixed, one can also choose the optimal method using a synthetic dataset processed with Norm-Sub. The intuition is that PowerNS improves over the approximate MLE (i.e., Norm-Sub, which is a theoretically testified method) by making the estimates closer to the underlying distribution. If one just want to estimate results for the most frequent values, one can use Norm. While Base can also be used, Norm reduces variance by utilizing the property that the estimates sum up to 1. These two methods do not change any value dramatically. Finally, if one cares about single value queries only, Base-Cut should be used. This is because when many values in the dataset are of low frequency, converting low estimates to 0 benefit the utility. Overall, one can follow the guideline for choosing post-processing methods.

- When single value queries are desired, use Base-Cut.
- When frequent values are desired, use Norm.
- When set-value queries are desired, use PowerNS or select one using synthetic datasets.

8.3 Related Work

There exist efforts to post-process results in the setting of centralized DP. Most of them focus on utilizing the structural information in problems other than the simple histogram, e.g., estimating marginals [28], [44] and hierarchy structure [17]. The methods do not consider the non-negativity constraint. Other than that, they are similar to Norm-Sub and minimize L_2 distance. On the other hand, the authors of [45] started from MLE and propose a method to minimize L_1 instead of L_2 distance, as the DP noise follows Laplace distribution.

In the LDP setting, Kairouz et al. [41] study exact MLE for GRR and RAPPOR [2]; and empirically show exact MLE performs worse than Norm-Sub. In [42], Bassily proves the error bound of Norm-Sub for the Hadamard Response mechanism. Jia et al. [43] propose to use external information about the dataset's distribution (e.g., assume the underlying dataset follows Gaussian or Zipf's distribution). We note that such information may not always be available. On the other hand, we exploit the basic information in each LDP setting. That is, first, the total number of users is known; second, negative values are not possible. We found that in the LDP setting, on the contrary to [41], minimizing L_2 distance achieves MLE under the approximation that the noise is close to the Gaussian distribution. There are also post-processing techniques proposed for other settings: Blasiok et al. [46] study the post-processing for linear queries, which generalizes histogram estimation; but their method only applied to a non-optimal LDP mechanism. [16] and [40] consider the hierarchy structure and apply the technique of [17]. [47] considers mean estimation and propose to project the result into $[0, 1]$.

9. PRIVACY AMPLIFICATION VIA SHUFFLING

(A version of this chapter has been previously published in VLDB 2020 [48].)

Recently, researchers introduced settings where one can achieve a middle ground between DP and LDP, in terms of both privacy and utility. This is achieved by introducing an additional party [49]–[52]. The setting is called the *shuffler model*. In this model, each user adds LDP noise to data, encrypts it, and then sends it to the new party called the shuffler. The shuffler permutes the users' reported data, and then sends them to the server. Finally the server decrypts the reports and obtains the result. In this process, the shuffler only knows which report comes from which user, but does not know the content. On the other hand, the server cannot link a user to a report because the reports are shuffled. The role of the shuffler is to break the linkage between the users and the reports. Intuitively, this anonymity can provide some privacy benefit. Therefore, users can add less noise while achieving the same level of privacy.

In this chapter, we study this new model from two perspectives. First, we examine from the algorithmic aspect, and make improvement to existing techniques. More specifically, in [51], it is shown the essence of the privacy benefit comes from a "noise" whose distribution is independent of the input value, also called privacy blanket. While existing work leverages this, it only works well when each user's value is drawn from a small domain. To obtain a similar privacy benefit when the domain is large, we propose to use the local hashing idea (see Chapter 3). That is, each user selects a random hash function, and uses LDP to report the hashed result, together with the selected hash function. By analyzing the utility and optimizing the parameters with respect to the utility metric (mean squared error), we present an algorithm that achieves accuracy orders of magnitude better than existing method. We call it Shuffler-Optimal Local Hash (SOLH).

We then work from the security aspect of the model. We review the system setting of this model and identify two types of attack that were overlooked: collusion attack and data-poisoning attack. Specifically, as there are more parties involved, there might exist collusions. While existing work assumes non-collusion, we explicitly consider the consequences of collusions among different parties and propose a protocol Private Encrypted Oblivious

Shuffle (PEOS) that is safer under these colluding scenarios. The other attack considers the setting where the additional party introduces calibrated noise to bias the result or break the privacy protection. To overcome this, our protocol PEOS takes advantage of cryptographic tools to prevent the shufflers from adding arbitrary noise.

9.1 Background

We briefly review the cryptographic primitives that will be used. Note that throughout this paper, we assume that the cryptographic tools are secure.

Additive Homomorphic Encryption. In Additive Homomorphic Encryption (AHE) [53], one can apply an algebraic operation (denoted by $\oplus$, e.g., multiplication) to two ciphertexts c_1, c_2, and get the ciphertext of the addition of the corresponding plaintexts. More formally, there are two functions, encrypt function Enc and decrypt function Dec. Given two ciphertexts $c_1 = \text{Enc}(v_1)$ and $c_2 = \text{Enc}(v_2)$, we have $c_1 \oplus c_2 = \text{Enc}(v_1 + v_2)$.

Additive Secret Sharing. In this technique, a user splits a secret value $v \in \{0, \ldots, d-1\}$ into $r > 1$ shares $\langle s_i \rangle_{i \in [r]}$, where $r - 1$ of them are randomly selected, and the last one is computed so that $\sum_i s_i \mod d = v$. The shares are then sent to r parties, so that each party only sees a random value, and v cannot be recovered unless all the r parties collaborate.

Oblivious Shuffle. In order to prevent the shuffler from knowing the mapping between the input and the output, oblivious shuffle introduces multiple shufflers. A natural method is to connect the shufflers sequentially; and each shuffler applies a random shuffle. Another way of achieving oblivious shuffle is the resharing-based shuffle [54], [55] which utilizes secret sharing. Suppose there are r shufflers. The users send their values to shufflers using secret sharing. Define $t = \lfloor r/2 \rfloor + 1$ as the number of "hiders", and $r - t$ as the number of "seekers". The resharing-based oblivious shuffle [55] proceeds like a "hide and seek" game. In particular, there are $\binom{r}{t}$ partitions of the r auxiliary servers into hiders and seekers. For each partition, the seekers each splits its vector of shares into t parts and sends them to the t hiders, respectively. Then the hiders accumulate the shares and shuffle their vectors using an agreed permutation. The shuffled vectors are then split into r shares and distributed to all of the r auxiliary servers. Note that now only the t hiders know the permutation order.

The process proceeds for $\binom{r}{t}$ rounds to ensure that none of the colluding $r-t$ auxiliary servers know about the final permutation order.

9.2 Summary of Existing Results

Throughout the chapter, we focus on the basic tool of histogram estimation.

The shuffling idea was originally proposed in Prochlo [56], where a shuffler is inserted between the users and the server to break the linkage between the report and the user identification. The privacy benefit was investigated in [49]–[51]. It is proven that when each user reports the private value using GRR with ϵ_l-LDP, applying shuffling ensures centralized (ϵ_c, δ)-DP, where $\epsilon_c < \epsilon_l$. Table 9.1 gives a summary of these results. Among them, [51] provides the strongest result in the sense that the ϵ_c is the smallest, and the proof technique can be applied to other LDP protocols.

Table 9.1. Privacy amplification result comparison. Each row corresponds to a method. The amplified ϵ_c only differs in constants. The circumstances under which the method can be used are different.

Method	Condition	ϵ_c
[50]	$\epsilon_l < 1/2$	$\sqrt{144\ln(1/\delta)\cdot\frac{\epsilon_l^2}{n}}$
[49]	$\sqrt{\frac{192}{n}\ln(4/\delta)} < \epsilon_c < 1$, binary	$\sqrt{32\ln(4/\delta)\cdot\frac{e^{\epsilon_l}+1}{n}}$
[51]	$\sqrt{\frac{14\ln(2/\delta)d}{n-1}} < \epsilon_c \leq 1$	$\sqrt{14\ln(2/\delta)\cdot\frac{e^{\epsilon_l}+d-1}{(n-1)}}$

Privacy Blanket. The technique used in [51] is called *blanket decomposition*. The idea is to decompose the probability distribution of an LDP report into two distributions, one dependent on the true value and the other independently random; and this independent distribution forms a "privacy blanket". In particular, the output distribution of GRR given in Equation (3.4) is decomposed into

$$\forall_{y\in D}\ \Pr[\mathsf{GRR}(v)=y] = (1-\gamma)\Pr^{v}[y] + \gamma\Pr[\mathsf{Uni}(D)=y]$$

where $\Pr^{v}[y]$ is the distribution that depends on v, and $\mathsf{Uni}(D)$ is uniformly random with $\Pr[\mathsf{Uni}(D)=y]=1/d$. With probability $1-\gamma$, the output is dependent on the true input; and

with probability γ, the output is random. Given n users, the $n-1$ (except the victim's) such random variables can be seen as containing some uniform noise (i.e., the $\gamma \Pr[\mathsf{Uni}(D) = y]$ part). For each value $v \in D$, the noise follows $\mathcal{B}(n-1, \gamma/d)$. Intuitively, this noise makes the output uncertain. The following theorem, which is derived from Theorem 3.1 of [51], formalizes this fact.

Theorem 9.2.1 (Binomial Mechanism)**.** *Binomial mechanism adds independent noise* $\mathcal{B}(n, p)$ *to each component of the histogram. It satisfies* (ϵ_c, δ)*-DP where*

$$\epsilon_c = \sqrt{\frac{14 \ln(2/\delta)}{np}}$$

In Theorem 9.2.1, the larger γ is, the better the privacy. Given GRR, we can maximize γ by setting $\Pr^v[y] = 1_{v=y}$, which gives us $\gamma = \frac{d}{e^{\epsilon_l}+d-1}$. The binomial noise $\mathcal{B}\left(n-1, \frac{1}{e^{\epsilon_l}+d-1}\right)$ thus provides $(\sqrt{14 \ln(2/\delta) \cdot \frac{e^{\epsilon_l}+d-1}{(n-1)}}, \delta)$-DP [51]. One limitation of [51] is that as GRR is used, the accuracy downgrades with domain size d.

Recent Results. Parallel to our work, [57], [58] propose mechanisms other than GRR to improve utility in this model. They both rely on the privacy blanket idea. The method in [57] gives better utility as it does not depend on $|D|$. However, the communication cost for each user is linear in $|D|$, which is undesirable when $|D|$ is large. Moreover, its accuracy is worse than the method proposed in our paper. We will analytically and empirically compare with [57].

9.3 Improving Utility of the Shuffler Model

In order to benefit from the shuffler model in the case when the domain size d is large, the key is to derive a mechanism whose utility does not degrade with d.

9.3.1 Unary Encoding for Shuffling

We first revisit the unary-encoding-based methods, also known as the basic RAPPOR [2], and show that this class of methods can enjoy the benefit of the privacy blanket argument. In particular, in unary-encoding, the value v is transformed into a vector B of size d, where

$B[v] = 1$ and the other locations of B are zeros (note that this requires values of the domain D be indexed from 1 to d). Then each bit b of B is perturbed to $1 - b$ independently. To satisfy LDP, the perturbation probability is set to $\frac{1}{e^{\epsilon/2}+1}$. Note that we use $\epsilon/2$ because for any two values v and v, their corresponding unary encodings differ by two bits. We can apply the privacy blanket argument and prove that a ϵ_l-LDP unary-encoding method satisfies (ϵ_c, δ)-DP after shuffling.

Theorem 9.3.1. *Given an ϵ_l-LDP unary-encoding method, after shuffling, the protocol is (ϵ_c, δ)-DP, where*

$$\epsilon_c = 2\sqrt{14\ln(4/\delta) \cdot \frac{e^{\epsilon_l/2}+1}{n-1}}$$

Proof: For any two neighboring datasets $D \simeq D$, w.l.o.g., we assume they differ in the n-th value, and $v_n = 1$ in D, $v_n = 2$ in D. By the independence of the bits, probabilities on other locations are equivalent. Thus we only need to examine the summation of bits for location 1 and 2. For each location, there are $n - 1$ users, each reporting the bit with probability

$$\forall_{y\in\{0,1\}} \Pr[B[j] \to y] = (1-\gamma)1_{B[j]=y} + \gamma \Pr[\mathsf{Uni}(2) = y]$$

where we slight abuse the notation and use $\mathsf{Uni}(2)$ for $\mathsf{Uni}(\{0, 1\})$. Given that the perturbation probability is $\Pr[1 \to 0] = \Pr[0 \to 1] = \frac{1}{e^{\epsilon_l/2}+1} = \gamma/2$, we can calculate that $\gamma = \frac{2}{e^{\epsilon_l/2}+1}$. After shuffling, the histogram of $n - 1$ (except the victim's) such random variables follows $\mathcal{B}(n - 1, \gamma/2)$. As there are two locations, by Theorem 9.2.1, we have $\epsilon_c = 2\sqrt{14\ln(4/\delta) \cdot \frac{e^{\epsilon_l/2}+1}{n-1}}$. □

9.3.2 Local Hashing for Shuffling

While sending B when d is large is fine for each user; with n users, receiving B's from the server side is less tolerable as it incurs $O(d \cdot n)$ bandwidth. To reduce the communication cost, we propose a hashing-based method, with a tradeoff between computation and commu-

nication. From the server side, it requires more computation cost than the unary-encoding based methods; but the overall communication bandwidth is smaller. In what follows, we prove the hashing-based method is private in the shuffler model.

We remind the readers that in local hashing, each user reports H and $y = \mathsf{GRR}(H(v))$. The hash function H is chosen randomly from a universal hash family and hashes v from a domain of size d into another domain of size $d' \leq d$; and GRR will report $H(v)$ with probability $\frac{e^{\epsilon_l}}{e^{\epsilon_l}+d'-1}$, and any other value (from the domain of size d') with probability $\frac{1}{e^{\epsilon_l}+d'-1}$ (Equation (3.4)). In terms of blanket decomposition, the user reports truthfully with probability $1-\gamma = \frac{e^{\epsilon_l}-1}{e^{\epsilon_l}+d'-1}$; and if the user reports randomly, any value from $[d']$ can be reported with equal probability. We call this method SOLH, which stands for Shuffler-Optimal Local Hash.

Theorem 9.3.2. *Given the ϵ_l-LDP SOLH method, after shuffling, the protocol is (ϵ_c, δ)-DP, where*

$$\epsilon_c = \sqrt{\frac{14\ln(2/\delta)(e^{\epsilon_l}+d'-1)}{n-1}}$$

Proof: Denote $\mathcal{A}$ as the algorithm of SOLH in the shuffler model. Let $[\langle H_i, y_i\rangle]_{i\in[n]}$ be the outputs of all users before shuffling, and let $[\langle \hat{H}_j, \hat{y}_j\rangle]_{j\in[n]}$ be the output of $\mathcal{A}(D)$. W.l.o.g., we assume D and D differ in the n-th value, i.e., $v_n \neq v_n$. We denote R as the output from $\mathcal{A}(D)$. To prove $\mathcal{A}$ is (ϵ_c, δ)-DP, it suffices to show

$$\mathsf{Pr}_{R\sim\mathcal{A}(D)}\left[\frac{\mathsf{Pr}\left[\mathcal{A}(D)=R\right]}{\mathsf{Pr}\left[\mathcal{A}(D)=R\right]} \geq e^{\epsilon_c}\right] \leq \delta$$

where the randomness is on coin tosses of all users' LDP mechanism and the shuffler's random shuffle. We first upper bound $\frac{\mathsf{Pr}[\mathcal{A}(D)=R]}{\mathsf{Pr}[\mathcal{A}(D)=R]}$ by assuming user n also report truthfully.

That is (we shorten the notation and use $\mathsf{Pr}\left[X(D)\right]$ to denote $\mathsf{Pr}\left[\mathcal{A}(D) = R\right]$),

$$\begin{aligned}
&\frac{\mathsf{Pr}\left[X(D)\right]}{\mathsf{Pr}\left[X(D)\right]} \\
=&\frac{\mathsf{Pr}\left[X(D) \mid \mathrm{Tru}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Tru}_n\right] + \mathsf{Pr}\left[X(D) \mid \mathrm{Rnd}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Rnd}_n\right]}{\mathsf{Pr}\left[X(D) \mid \mathrm{Tru}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Tru}_n\right] + \mathsf{Pr}\left[X(D) \mid \mathrm{Rnd}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Rnd}_n\right]} \\
=&\frac{\mathsf{Pr}\left[\mathsf{Pr}\left[X(D)\right] \mid \mathrm{Tru}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Tru}_n\right] + c}{\mathsf{Pr}\left[X(D) \mid \mathrm{Tru}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Tru}_n\right] + c} \leq \frac{\mathsf{Pr}\left[\mathsf{Pr}\left[X(D)\right] \mid \mathrm{Tru}_n\right]}{\mathsf{Pr}\left[X(D) \mid \mathrm{Tru}_n\right]}
\end{aligned}$$

where $c = \mathsf{Pr}\left[X(D) \mid \mathrm{Rnd}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Rnd}_n\right] = \mathsf{Pr}\left[X(D) \mid \mathrm{Rnd}_n\right] \cdot \mathsf{Pr}\left[\mathrm{Rnd}_n\right]$ is a constant. Thus we assume user n reports truthfully, and omit the conditional part for simplicity. The rest of the proof proceeds in 5 steps:

• *Step 1 (expand the probability expression):*

Denote T as indices of the first $n-1$ users who report truthfully (i.e., with probability $1-\gamma = \frac{e^{\epsilon_l}-1}{e^{\epsilon_l}+d'-1}$), and let R_T denote their chosen hash functions and hashed results ($R_T = [\langle H_i, y_i\rangle]_{i\in T}$). We examine the conditional probability $\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right]$:

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right] = \sum_{\boldsymbol{\pi}} \mathsf{Pr}\left[\boldsymbol{\pi}\right] \mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T, \boldsymbol{\pi})\right] \\
=&\sum_{\boldsymbol{\pi}} \mathsf{Pr}\left[\boldsymbol{\pi}\right] \left(\underbrace{\prod_{i\in T} \mathsf{Pr}\left[H_{\boldsymbol{\pi}(i)}\right] 1_{H_{\boldsymbol{\pi}(i)}=\hat{H}_i \wedge y_{\boldsymbol{\pi}(i)}=\hat{y}_i}}_{\text{reports from users in } T} \cdot \underbrace{\prod_{i\in[n-1]\setminus T} \mathsf{Pr}\left[H_{\boldsymbol{\pi}(i)}\right] \frac{1}{d'}}_{\text{reports from users in } [n-1]\setminus T} \cdot \underbrace{\mathsf{Pr}\left[H_{\boldsymbol{\pi}(n)}\right] 1_{H_{\boldsymbol{\pi}(n)}(v_n)=y_{\boldsymbol{\pi}(n)}}}_{\text{report from user } n} \right)
\end{aligned} \tag{9.1}$$

$\mathsf{Pr}\left[\boldsymbol{\pi}\right]$ denotes the probability a specific random permutation is chosen ($\mathsf{Pr}\left[\boldsymbol{\pi}\right] = 1/n!$), $\mathsf{Pr}\left[H_{\boldsymbol{\pi}(i)}\right]$ (short for $\mathsf{Pr}\left[\hat{H}_i = H_{\boldsymbol{\pi}(i)}\right]$) is the probability user i chooses hash function $H_{\boldsymbol{\pi}(i)}$ (assuming there are h possible hash functions, $\mathsf{Pr}\left[H_{\boldsymbol{\pi}(i)}\right] = 1/h$), and the summation is over all permutation $\boldsymbol{\pi}$. Users are divided into three groups. For $i \in T$, we know from R_T that his/her report is $\langle \hat{H}_i, \hat{y}_i\rangle$, and it must match $\langle H_{\boldsymbol{\pi}(i)}, y_{\boldsymbol{\pi}(i)}\rangle$ (otherwise $\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T, \boldsymbol{\pi})\right] = 0$). We use the indicator function to denote this. Here as the user reports truthfully, $\hat{y}_i = \hat{H}_i(v_i)$, and $1_{H_{\boldsymbol{\pi}(i)}=\hat{H}_i \wedge y_{\boldsymbol{\pi}(i)}=\hat{y}_i} = 1_{H_{\boldsymbol{\pi}(i)}=\hat{H}_i \wedge H_{\boldsymbol{\pi}(i)}(v_i)=y_{\boldsymbol{\pi}(i)}}$. For user n and users who report randomly, their probabilities can also be analyzed similarly.

• *Step 2 (convert probabilities to counts):*

Denote $P = \{\boldsymbol{\pi} \mid \forall i \in T, H_{\pi(i)} = \hat{H}_i \wedge y_{\pi(i)} = \hat{y}_i\}$. Here P is the set of all possible permutations that make the $i \in T$ part of Equation (9.1) non-zero (i.e., all the indicator functions for $i \in T$ equal 1). Assuming the reports in R are distinct (i.e., $\nexists i, j \in [n]$ s.t. $H_i = H_j \wedge y_i = y_j$), such permutations must map $i \in T$ to $\boldsymbol{\pi}(i)$ s.t. $H_{\pi(i)} = \hat{H}_i$ and $y_{\pi(i)} = \hat{y}_i$. P can be partitioned into $n - |T|$ equal-sized subsets each with $\boldsymbol{\pi}(n) = i$. That is, for each $i \in [n] \setminus T$, define $P_i = \{\boldsymbol{\pi} \mid \boldsymbol{\pi} \in P \wedge \boldsymbol{\pi}(n) = i\}$. Each P_i is of size $\mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \cdot (n-1-|T|)!$ because P_i left the mapping of $[n-1] \setminus T$ unspecified (and any random permutation is possible). We now have:

$$\begin{aligned}
&\frac{\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right]}{\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right]} = \frac{c_1 \sum_{\boldsymbol{\pi} \in P} \mathbf{1}_{H_{\pi(n)}(v_n)=y_{\pi(n)}}}{c_1 \sum_{\boldsymbol{\pi} \in P} \mathbf{1}_{H_{\pi(n)}(v_n)=y_{\pi(n)}}} \\
=&\frac{\sum_{i \in [n]\setminus T} \sum_{\boldsymbol{\pi} \in P_i} \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i}}{\sum_{i \in [n]\setminus T} \sum_{\boldsymbol{\pi} \in P_i} \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i}} = \frac{\sum_{i \in [n]\setminus T} \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i}}{\sum_{i \in [n]\setminus T} \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i}}
\end{aligned} \tag{9.2}$$

where $c_1 = \mathsf{Pr}\left[\boldsymbol{\pi}\right] \left(\prod_{i \in [n]} \mathsf{Pr}\left[H_{\pi(i)}\right]\right)\left(\prod_{i \in [n-1]\setminus T} \frac{1}{d'}\right)$ is a constant that does not depend on v_n or v_n. Note that we previously assumed the reports in R are unique. If there are duplicated reports, P could be larger, but the ratio stays the same.

To see this, define $R_{-T} = [\langle \hat{H}_i, \hat{y}_i \rangle]_{i \in [n]\setminus T}$ as reports from $[n] \setminus T$. We model a valid permutation in P as a two-step process: for any report from user $i \in [n] \setminus T$, suppose there are $a_i \geq 0$ reports in R_T that is the same (both the hash function and the hash result are same) as user i's report, and $b_i \geq 1$ duplicated reports in R_{-T}. We first choose a_i from $a_i + b_i$ reports and "put" them to R_T; then we permute R_T (there are $c \geq 1$ valid permutations within R_T) and R_{-T} (there are $\sum_{i \in [n]\setminus T} \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \cdot (n-1-|T|)!$ valid permutations in R_{-T}). It can be verified that this modeling covers exactly all permutations in P. Now for each $i \in [n] \setminus T$: If $a_i = 0$, there are $x_i = \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \cdot \prod_{i \in [n]\setminus T} \binom{a_i+b_i}{a_i} \cdot c \cdot (n-1-|T|)!$ possible permutations in P, where $\prod_{i \in [n]\setminus T} \binom{a_i+b_i}{a_i}$ denotes the number of possible choices for the duplicated reports. If $a_i > 0$, denote $y_i = x_i / \binom{a_i+b_i}{a_i}$. We consider all these $a_i + b_i$ duplicate reports together. Index n can be mapped to match any of the $a_i + b_i$ duplicated reports. For each report, there are $\binom{a_i+b_i-1}{a_i}$ choices (because the permutation will first choose a_i reports and put them into R_T, and the current report which n is mapped to cannot be put to R_T; thus we choose a_i from the remaining $a_i + b_i - 1$ reports to put to R_T). Overall, we have

$y_i \cdot (a_i + b_i) \cdot \binom{a_i+b_i-1}{a_i} = y_i \cdot b_i \cdot \binom{a_i+b_i}{a_i} = x_i \cdot b_i$ valid permutations, which equals to the case when we sum all the b_i values each with x_i permutations. Therefore, there are $x_i = \mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \cdot c$ valid permutations for each $i \in [n] \setminus T$. Summarizing all x_i's gives us Equation (9.2).

• *Step 3 (model the counts with Binomial RVs):*

So far, we have proved that, fixing R, T and R_T, the ratio only depends on the numbers of reports that are random and matches v_n and v_n, respectively. The high level idea is to show that knowing T and R_T fixes the permutation on values from T; and any valid permutation only shuffles values from $[n] \setminus T$ (informally, this can be thought of as the server removes reports from T). Now define

$$N_{R,T,R_T} = \sum_{i \in [n] \setminus T} \left(\mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \right) \text{ and } N_{R,T,R_T} = \sum_{i \in [n] \setminus T} \left(\mathbf{1}_{\hat{H}_i(v_n)=\hat{y}_i} \right),$$

we want to prove

$$\begin{aligned}
&\mathsf{Pr}_{(R,T,R_T) \sim \mathcal{A}(D)} \left[\frac{\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right]}{\mathsf{Pr}\left[\mathcal{A}(D) = R \mid (T, R_T)\right]} \geq e^{\epsilon_c} \right] \\
&\text{(omit the } (R, T, R_T) \sim \mathcal{A}(D) \text{ part to simplify notations)} \\
=&\mathsf{Pr}\left[\frac{N_{R,T,R_T}}{N_{R,T,R_T}} \geq e^{\epsilon_c} \right] \\
\leq& 1 - \mathsf{Pr}\left[N_{R,T,R_T} \leq \theta e^{\epsilon_c/2} \wedge N_{R,T,R_T} \geq \theta e^{-\epsilon_c/2} \right] \\
\leq& \mathsf{Pr}\left[N_{R,T,R_T} \geq \theta e^{\epsilon_c/2} \right] + \mathsf{Pr}\left[N_{R,T,R_T} \leq \theta e^{-\epsilon_c/2} \right]
\end{aligned}$$

where θ is some constant. For (R, T, R_T) generated from a random run of $\mathcal{A}(D)$, we can show N_{R,T,R_T} and N_{R,T,R_T} follow Binomial distributions. In particular, as we assumed user n always report truth, there must be $H_n(v_n) = y_n$; the remaining $n-1$ users will first decide whether to report truthfully (i.e., with probability $(e^{\epsilon_l}-1)/(e^{\epsilon_l}+d'-1)$), and if user i's report $\langle H_i, y_i \rangle$ is random, we have $\mathsf{Pr}\left[H_i(v_n) = y_i\right] = 1/d'$. Each user's reporting process are thus modeled as two Bernoulli processes. As a result, N_{R,T,R_T} follows the Binomial distribution $\mathcal{B}\left(n-1, 1/(e^{\epsilon_l}+d'-1)\right)$ plus a constant 1. Similarly, $N_{R,T,R_T} \sim \mathcal{B}\left(n-1, 1/(e^{\epsilon_l}+d'-1)\right) + \mathbf{1}_{H_n(v_n)=y_n} \geq \mathcal{B}\left(n-1, 1/(e^{\epsilon_l}+d'-1)\right)$.

• *Step 4 (bound the ratio of Binomials with Chernoff bounds):*

Following the later part of the proof of Theorem 3.1 from [51] : set $\theta = \frac{n-1}{e^{\epsilon_l}+d'-1} = \mathrm{E}\left[N_{R,T,R_T} \right] = \frac{14\log(2/\delta)}{\epsilon^2}$,

$$
\begin{aligned}
&\mathsf{Pr}\left[N_{R,T,R_T} \geq \theta e^{\epsilon_c/2}\right] + \mathsf{Pr}\left[N_{R,T,R_T} \leq \theta e^{-\epsilon_c/2}\right] \\
=&\mathsf{Pr}\left[N_{R,T,R_T} \geq \theta e^{\epsilon_c/2} - 1\right] + \mathsf{Pr}\left[N_{R,T,R_T} \leq \theta e^{-\epsilon_c/2}\right] \\
\leq&\mathsf{Pr}\left[N_{R,T,R_T} - \mathrm{E}\left[N_{R,T,R_T} \right] \geq \theta(e^{\epsilon/2} - 1 - 1/\theta)\right] \\
+&\mathsf{Pr}\left[N_{R,T,R_T} - \mathrm{E}\left[N_{R,T,R_T} \right] \leq \theta(e^{-\epsilon/2} - 1)\right] \\
\leq&\exp(-\theta(e^{\epsilon/2} - 1 - 1/\theta)^2/3) + \exp(-\theta(1 - e^{-\epsilon/2})^2/2)
\end{aligned}
$$

Assuming $\epsilon \leq 1$, both of them are less than or equal to $\delta/2$: For the first term, $\theta \geq \frac{27}{\epsilon}$ implies $e^{\epsilon/2} - 1 - 1/\theta \geq \frac{25}{54}\epsilon$ and $14 \geq \frac{3 \cdot 54^2}{25^2}$. For the second term, $1 - e^{\epsilon/2} \geq (1 - e^{1/2})\epsilon \geq \epsilon/\sqrt{7}$.

• *Step 5 (put things together):*

We have bound the conditional probability ratio. It also implies a bound on joint probability ratio, because $\frac{\mathsf{Pr}[\mathcal{A}(D)=R|(T,R_T)]}{\mathsf{Pr}[\mathcal{A}(D)=R|(T,R_T)]} = \frac{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]\mathsf{Pr}[T,R_T]}{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]\mathsf{Pr}[T,R_T]} = \frac{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]}{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]}$. For any R, we say (T, R_T) is "good" if $e^{\epsilon_c} \geq \frac{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]}{\mathsf{Pr}[\mathcal{A}(D)=R\wedge(T,R_T)]}$ and "bad" otherwise. Consider any possible set S of output, we finally prove

$$
\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A}(D) \in S\right] = \sum_{(T,R_T)} \sum_{R\in S} \mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] \\
=& \sum_{(T,R_T) \text{ is good}} \sum_{R\in S} \mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] \\
+& \sum_{(T,R_T) \text{ is bad}} \sum_{R\in S} \mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] \\
\leq& \sum_{(T,R_T) \text{ is good}} \sum_{R\in S} e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] \\
+& \sum_{(T,R_T) \text{ is bad}} \sum_{R} \mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] \\
\leq& \sum_{(T,R_T)} \sum_{R\in S} e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}(D) = R \wedge (T, R_T)\right] + \delta \\
=&e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}(D) \in S\right] + \delta
\end{aligned}
$$

□

9.3.3 Utility Analysis

Now we analyze the utility of different methods. We utilize the pure framework from Chapter 3 to analyze the accuracy of estimating the frequency of each value in the domain. In particular, we measure the expected squared error of the estimation $\tilde{f}_v$, which equals variance, i.e.,

$$\sum_{v \in D} \mathrm{E}\left[(\tilde{f}_v - f_v)^2 \right] = \sum_{v \in D} \mathrm{Var}\left[\tilde{f}_v \right]$$

Fixing the local ϵ_l, the variances are already summarized in Chapter 3; our analysis extends that into the shuffler setting. We fix ϵ_c and estimate variance for different methods.

Utility of Generalized Randomize Response. We first prove the variance of GRR.

Proposition 9.3.3. *Given ϵ_c in the shuffler model, the variance of using* GRR *is bounded by* $\frac{\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-1}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-d\right)^2}$.

Proof: Given the domain size d and the LDP parameter ϵ_l, the variance is given in Chapter 3. Here for completeness, we present the full proof. We will omit these steps in the following proofs. Denote $p = \frac{e^{\epsilon_l}}{e^{\epsilon_l}+d-1}$, $q = \frac{1}{e^{\epsilon_l}+d-1}$, and y_i is the report of user i, we have

$$\mathrm{Var}\left[\tilde{f}_v \right] = \mathrm{Var}\left[\frac{1}{n} \left(\sum_{i \in [n]} \frac{\mathbb{1}_{v=y_i} - q}{p-q} \right) \right] = \frac{1}{n^2} \mathrm{Var}\left[\sum_{i \in [n]} \frac{\mathbb{1}_{v=y_i}}{p-q} \right] = \frac{\sum_{i \in [n]} \mathrm{Var}\left[\mathbb{1}_{v=y_i} \right]}{n^2 \cdot (p-q)^2}$$

Here for each of the n users, if the true value is v (there are nf_v of them) we have $\mathrm{Var}\left[\mathbb{1}_{v=y_i} \right] = p(1-p)$; otherwise, we have $\mathrm{Var}\left[\mathbb{1}_{v=y_i} \right] = q(1-q)$ for the rest $n(1-f_v)$ users. Together, we have

$$\mathrm{Var}\left[\tilde{f}_v \right] = \frac{nf_v p(1-p) + n(1-f_v)q\,(1-q)}{n^2(p-q)^2} = \frac{q(1-q)}{n(p-q)^2} + \frac{f_v\,(1-p-q)}{n(p-q)}$$

Plugging in the value of p and q, and assuming f_v is small on average, then we have

$$\mathrm{Var}\left[\tilde{f}_v\right] \leq \frac{q(1-q)}{n(p-q)^2} = \frac{e^{\epsilon_l}+d-2}{n(e^{\epsilon_l}-1)^2}$$

From [51], we have $e^{\epsilon_l}+d-1 = \frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}$. Thus the variance becomes $\frac{\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-1}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-d\right)^2}$. □

Utility of Unary Encoding (RAPPOR). Similarly, we can prove the variance of unary encoding.

Proposition 9.3.4. *Given ϵ_c in the shuffler model, the variance of using unary encoding (RAPPOR) is bounded by* $\frac{\frac{\epsilon_c^2(n-1)}{56\ln(4/\delta)}-1}{n\left(\frac{\epsilon_c^2(n-1)}{56\ln(4/\delta)}-2\right)^2}$.

Proof: According to [5], the variance of RAPPOR given ϵ_l is

$$\frac{e^{\epsilon_l/2}}{n(e^{\epsilon_l/2}-1)^2}$$

From Theorem 9.3.1, we have $e^{\epsilon_l/2}+1 = \frac{\epsilon_c^2(n-1)}{56\ln(4/\delta)}$. Thus the variance becomes $\frac{\frac{\epsilon_c^2(n-1)}{56\ln(4/\delta)}-1}{n\left(\frac{\epsilon_c^2(n-1)}{56\ln(4/\delta)}-2\right)^2}$. □

Utility of Local Hashing. Now we prove the variance of SOLH and instantiate d'.

Proposition 9.3.5. *Given ϵ_c in the shuffler model, the variance of using* SOLH *is bounded by* $\frac{\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}\right)^2}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}-d'\right)^2(d'-1)}$.

Proof: According to Equation (10) of [5], the variance of local hashing given ϵ_l is

$$\frac{(e^{\epsilon_l}+d'-1)^2}{n(e^{\epsilon_l}-1)^2(d'-1)} \tag{9.3}$$

From Theorem 9.3.2, we have $e^{\epsilon_l} + d' - 1 = \frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}$. Thus the variance is $\frac{\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}\right)^2}{n\left(\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)} - d'\right)^2(d'-1)}$. □

Optimizing Local Hashing. Note that d' is unspecified. We can tune d' to optimize variance given a fixed ϵ_c. Denote m as $\frac{\epsilon_c^2(n-1)}{14\ln(2/\delta)}$, our goal is to choose d' that minimize this variance $\mathrm{Var}(m, d') = \frac{m^2}{n(m-d')^2(d'-1)}$. By making its partial derivative to 0, we can obtain that when

$$d' = \frac{m+2}{3} = \frac{\epsilon_c^2(n-1)}{42\ln(2/\delta)} + \frac{2}{3} \tag{9.4}$$

the variance is minimized. Note that d' can only be an integer. In the actual implementation, we choose d' to be $\lfloor (m+2)/3 \rfloor$. Thus the variance is optimized to $\mathrm{Var}(m, \lfloor (m+2)/3 \rfloor)$.

Comparison of the Methods. We first observe that the variance of GRR grows with d (as shown in Proposition 9.3.3). When d is large, we should use unary encoding or local hashing. Between the two, the variance of unary encoding is slightly better, however, its communication cost is higher. Thus, between GRR and SOLH, we can choose the one with better utility by comparing Proposition 9.3.3 and $\mathrm{Var}(m, \lfloor (m+2)/3 \rfloor)$.

9.3.4 Comparison with Parallel Work

Parallel to our work, [57], [58] also propose mechanisms to improve utility in this model. Among them [57] gives better utility which does not depend on $|D|$. Similar to our method, its proof also utilizes Theorem 9.2.1. But the approach is different. In particular, [57] first transforms the data using one-hot encoding, then independently increment values in each location with probability $p = 1 - \frac{200}{\epsilon_c^2 n}\ln(4/\delta)$. We call this method AUE for appended unary encoding. As each location is essentially a Bernoulli bit, its variance is $p(1-p) = \frac{200}{\epsilon_c^2 n}\ln(4/\delta)\left(1 - \frac{200}{\epsilon_c^2 n}\ln(4/\delta)\right)$. Compared with Lemma 9.3.5, this gives comparable results (differing by only a constant). But this protocol itself is not LDP. Moreover, as one-hot encoding is used, the communication cost for each user is linear in $|D|$, which is even worse than GRR. We will empirically compare with [57] in the experimental evaluation section.

More recently, [59] also proposed a similar unary-encoding-based method. We note that [59] operate on a novel removal LDP notion. More specifically, previous (ours included) LDP and shuffler-based LDP literature works with Definition 7.1.1, which ensures that for each user, if his/her value changes, the report distribution is similar. [59] introduces a novel removal LDP notion inspired by the removal DP. In particular, removal DP states that for any two datasets D and D_-, where D_- is obtained by removing any one record from D, the output distributions are similar. Extending that idea to the local setting, removal LDP states that for each user, whether his/her value is empty or not, the report distribution is similar. Given that, a unary-encoding-based method similar to RAPPOR [2] is proposed. The method is similar to the method we described in Section 9.3.1, except that privacy budget ϵ_l is not divided by 2. Interestingly, any ϵ-Removal LDP algorithm is also a 2ϵ-Replacement LDP algorithm, because

$$\mathsf{Pr}\left[\mathcal{A}(v) \in T\right] \leq e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}(\perp) \in T\right] \leq e^{2\epsilon}\mathsf{Pr}\left[\mathcal{A}(v) \in T\right]$$

where $\perp$ is a special "empty" input. As a result, in our LDP setting, the two methods achieves the same utility.

9.4 Security Analysis

This section focuses on the analyzing the security implications of the shuffler model. We identify different parties and potential attacks. Then we propose countermeasures using secret sharing and oblivious shuffle in next section.

9.4.1 Parties and Attackers

There are three types of parties in the shuffler model: *users*, the *server*, and the *auxiliary servers* (shufflers). The auxiliary servers do not exist in the traditional models of DP and

LDP; and in DP, the server may share result with some external parties. Figure 9.1 provides an overview of the system model.

The Attackers. From the point of view of a single user, other parties, including the auxiliary server, the server, and other users, could all be adversaries. We assume all parties have the same level of background knowledge, i.e., all other users' information except the victim's. This assumption essentially enables us to argue DP-like guarantee for each party.

The prominent adversary is the server. Other parties can also be adversaries but are not the focus because they have less information. For example, in the shuffler-based approach, there is only one auxiliary server. It knows nothing from the ciphertext.

Additional Threat of Collusion. We note that in the multi-party setting, one needs to consider the consequences when different parties collude. In general, there are many combinations of colluding parties. And understanding these scenarios enables us to better analyze and compare different approaches.

In particular, the server can collude with the auxiliary servers. If all the auxiliary servers are compromised, the model is reduced to that for LDP. Additionally, the server can also collude with other users (except the victim), but in this case the model is still LDP. On the other hand, if the server only colludes with other users, it is less clear how the privacy guarantee will downgrade. Other combinations are possible but less severe. Specifically, there is no benefit if the auxiliary servers collude with the users. We consider all potential collusions and highlight three important (sets of) adversaries:

- Adv: the server itself.
- Adv_u: the server colluding with other users.
- Adv_a: the server with the auxiliary servers.

9.4.2 Privacy Guarantees of Existing Methods

Having identified the potential adversaries and the proving technique, now we examine the shuffler-based DP. The key ideas are (1) We model each attack's view using an algorithm,

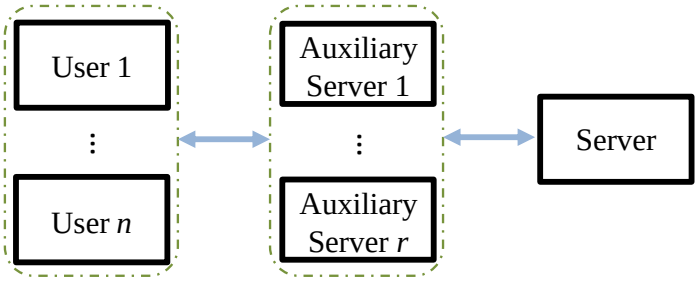


Figure 9.1. Overview of parties and interactions. Users communicate with the auxiliary servers. The auxiliary servers processes the users' data, and communicate with the server.

such that we can prove the DP guarantee. (2) We prove the DP guarantee for each party separately. Existing work focuses on Adv, but we examine the privacy guarantee against each of the Adv's. This gives a comprehensive understanding of the system's privacy guarantee.

In particular, existing work showed that if each user executes an ϵ_l-LDP protocol, the view of Adv is (ϵ_c, δ)-DP. If the users collude with the server, the server's view is composed of two parts: the shuffled reports as in Adv, and all users' reports except the victim's. By subtracting each user's reports from the shuffled result, the server now knows the victim's LDP report; thus the model falls back to the original setting. Finally, if the shuffler colludes with the server, the model also degrade to the LDP setting.

Note that we assume the cryptographic primitives are safe (i.e., the adversaries are computationally bounded and cannot learn any information from the ciphertext) and there are no side channels such as timing information. In some cases, the whole procedure can be interactive, i.e., some part of the observation may depend on what the party sends out. For this, one can utilize composition theorems to prove the DP guarantee. Moreover, the parties are assumed to follow the protocol in the privacy proofs. If the parties deviate from the prescribed procedure, we examine the possible deviations and their influences in the next subsection.

9.4.3 Robustness to Malicious Parties

There could be multiple reasons for each party to be malicious to (1) interrupt the data collection process, (2) infer more sensitive information from the users, and (3) degrade the utility (estimation accuracy) of the server. In what follows, for each of the reasons, we analyze the consequence and potential mitigation of different parties. Note that the server will not deviate from the protocol as it is the initiator, unless to infer more information of the users.

First, any party can try to interrupt the process; but it is easy to mitigate. If a user blocks the protocol, his report can be ignored. If the auxiliary server denies the service, the server can find another auxiliary server and redo the protocol. Note that in this case, users need to remember their report to avoid averaging attacks.

Second, it is possible that the auxiliary server deviates from the protocol (e.g., by not shuffling LDP reports), thus the server has access to the raw LDP reports. In these cases, the server can learn more information, but the auxiliary server does not have benefits except saving some computational power. And if the auxiliary server colludes with the server, they can learn more information without any deviation. Thus we assume the auxiliary server will not deviate in order to infer sensitive information. For the server, as it only sees and evaluates the final reports; and the reports are protected by LDP, there is nothing the server can do to obtain more information from the users.

Third, we note that any party can degrade the utility. Any party other than the server has the incentive to do this. For example, when the server is interested in learning the popularity of websites, different parties can deviate to promote some targeted website. This is also called the data poisoning attack. To do this, the most straight-forward way is to generate many fake users, and let them join the data collection process. This kind of Sybil attack is hard to defend against without some kind of authentication, which is orthogonal to the focus of this paper. Each user can change the original value or register fake accounts; and this cannot be avoided. But any ability beyond it is undesirable. In addition, the protocol should restrict the impact of the auxiliary server on the result.

To summarize, different parties can deviate from the protocol, but we argue that in most cases, a reasonable party has no incentive to do this, other than poisoning the result. We are mainly concerned about the users or the auxiliary server disrupting utility.

9.4.4 Discussion and Key Observations

In this section, we first systematically analyze the setting of the shuffler-based DP model. In addition to the adversary of the server, we highlight two more sets of adversaries. We then propose to analyze the privacy guarantee against different (sets of) adversaries. Finally, we discuss the potential concern of malicious parties. Several observations and lessons are worth noting.

When Auxiliary Server Colludes: No Amplification. When the server colludes with the auxiliary servers, the privacy guarantee falls back to the original LDP model. When using the shuffler model, we need to reduce the possibility of this collusion, e.g., by introducing more auxiliary servers.

When Users Collude: Possibility Missed by Previous Literature. When proving privacy guarantees against the server, existing work assumes the adversary has access to users' sensitive values but not the LDP output. While this is possible, we note that if an adversary already obtains users' sensitive values, it may also have access to the users' LDP reports. Such cases include the users (except the victim) collude with the server; or the server is controlling the users (except the victim). Thus, the assumption in the shuffle-based amplification work uncommon in real-world scenarios, which makes the privacy guarantee less intuitive to argue.

When Parties Deviates: Avoid Utility Disruption. The protocol should be designed so that each individual user or auxiliary server has limited impact on the estimation result.

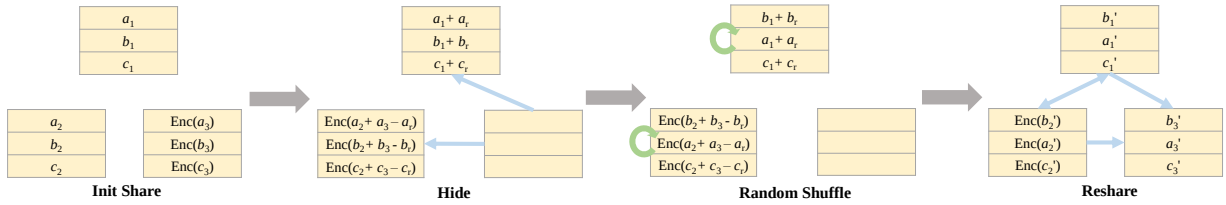


Figure 9.2. Overview of EOS with $r = 3$ shufflers and $n = 3$ values a, b, c. Each shuffler receives n shares; and one shuffler's shares are encrypted by additive homomorphic encryption. During hiding, one shuffler sends its shares to the other two shufflers, who then shuffle the aggregated shares with an agreed permutation. To reshare, each of the shufflers splits its shares and send them to the other shufflers.

9.5 Defending against Attacks

We present a protocol that improves the security guarantee of existing work. The goal is to simultaneously defend against three threats: (1) the server colludes with the users; (2) the server colludes with the auxiliary servers; (3) data poisoning from each party.

9.5.1 Fake Response from Auxiliary Servers

To defend against the threat when the server colludes with the users, we propose to have the auxiliary servers inject noise. There can be different ways to do this. Our approach utilizes uniform fake reports. The intuition of this approach is that (1) its analysis is compatible with the privacy blanket argument, which will be more clear later; and (2) the expected noise for each value in the domain is the same, thus suitable for obtaining a good privacy amplification effect. On the server side, after obtaining the estimated frequency $\tilde{f}$, the server recovers the frequency for the original dataset by subtracting the expected noise, i.e.,

$$f_v = \frac{n + n_r}{n} \tilde{f}_v - \frac{n_r}{n} \frac{1}{d} \tag{9.5}$$

Building on top of this, we present efforts to defend against the other two threats, i.e., the server colluding with the auxiliary servers, and data poisoning attack.

First Attempt: Sequential Shuffle

To improve the trust model of the shuffler-based model, one idea is to introduce a sequence of shufflers, so that as long as one shuffler is trusted, the privacy guarantee remains. In this case, the task of inserting n_r fake reports can be divided equally among the r auxiliary servers (shufflers). More specifically, the first shuffler receives the users' LDP reports as input, and draws $n_u = n_r/r$ fake reports. It then shuffles all the reports and sends them to the second shuffler, who draws another n_u fake reports, shuffles all the reports, and sends them to the next shuffler. This procedure proceeds until the last shuffler sends the result to the server. Onion encryption is used during the process; each party decrypts one layer of encryption, and the server obtains $n + n_r$ reports.

However, this approach is vulnerable to poison attacks by the shufflers. That is, the auxiliary servers can replace the users' reports with any report of their choice to change the final result, and the fake reports each shuffler inserts can be chosen arbitrarily.

To mitigate the first threat, we can use an idea of spot-checking. That is, the server can add dummy accounts before the system setup, then it can check whether the reports from his accounts are tampered. For the second threat, we find that it hard to handle. Specifically, a dishonest auxiliary server may draw fake reports from some skewed (instead of uniform) distribution in order to mislead the analyzer and achieve a desired result; and there is no way to self-prove the randomness he used is truly random.

Second Attempt: Oblivious Shuffle

To overcome the data poisoning attack, our approach is to construct the fake reports using secret sharing, which ensures that as long as one shuffler is honest, the inserted fake reports are uniformly random. To share an LDP report, we note that for both GRR and SOLH, the domain of the report can be mapped to an ordinal group $\{0, 1, \ldots, x\}$, where each index represents one different LDP report. Thus the LDP reports can be treated as numbers and shared with additive secret sharing.

In order to shuffle shares of secret, we utilize the oblivious shuffle protocol described in Section 9.1. More specifically, the n users each splits his/her LDP reports into r shares

among the r shufflers. Each of the shufflers then uniformly draws one share for each of the n_r fake reports. Thus the shufflers each has $n + n_r$ shares; and the sums of the shares equal to the n reports from users and n_r report that are random. An oblivious shuffle protocol is then executed among the shufflers to shuffle the $n + n_r$ shares of reports. Finally the r shufflers send their shares to the server, who combines the shares to obtain the results. Note that the communication is assumed to be done via secure channels.

This solution suffers from a threat that, even without the server, half of the shufflers can collude to recover the user reports. To mitigate this concern, we design a new oblivious shuffle protocol EOS that uses additive homomorphic encryption (AHE).

Proposal: Private Encrypted Oblivious Shuffle

To ensure that the shufflers cannot infer the users' reported data, a natural solution is to encrypt the shares using the server's public key. Moreover, the encryption needs to be additively homomorphic in order to be compatible with the secret-sharing operations. In what follows, we present a new protocol Encrypted Oblivious Shuffle (EOS) that utilizes additive homomorphic encryption (AHE) in oblivious shuffle. We then present our proposal Private Encrypted Oblivious Shuffle (shorted for PEOS) that uses EOS for DP.

Encrypted Oblivious Shuffle. Encrypted Oblivious Shuffle (EOS) works similarly to oblivious shuffle. One difference is that in each round, one shuffler will possess the encrypted shares. The encrypted shares can be shuffled and randomized just like normal shares except that they are then processed under AHE.

Denote the shuffler who possess encrypted shares as E. In each round, E splits its encrypted vector of shares into t new vectors so that $t - 1$ of which are plaintexts, and the last one is still in the ciphertext form (this can be done because of AHE). The t shares are randomly sent to the t hiders. Only one of them will receive the ciphertext share and become the next E. After the group shuffling, the new E splits its vector of shares and sends them to r parties. An example of EOS with $r = 3$ is demonstrated in Figure 9.2. EOS strengthens oblivious shuffle in that even if the r shufflers collude, they cannot figure out the users' original reports, because one share is encrypted.

Note that there is a crucial requirement for the AHE scheme: it should support a plaintext space of $\mathbb{Z}_{2^\ell}$ where ℓ is normally 32 or 64 in our case. This is because the fake reports are sampled locally as random ℓ-bit shares, and later they will be encrypted and added in AHE form, so that the decrypted result modulo 2^ℓ looks like other reports. Otherwise the fakeness will be detected by the server. Such an AHE scheme can be instantiated to be the full-decryption variant of DGK [60] using Pohlig-Hellman algorithm [61].

Corollary 9.5.1. *Encrypted oblivious shuffle, instantiated with additive homomorphic encryption of plaintext space $\mathbb{Z}_{2^\ell}$, is a secure oblivious shuffle protocol in the semi-honest model.*

Proof Sketch: The difference of EOS from oblivious shuffle is that AHE is used for one hider's computation in each round. As long as AHE does not leak additional information, similar proof about the final shuffling order can be derived from oblivious shuffle [55].

For AHE, note that although we use AHE for one hider's computation in each round, the computation is translated into modulo 2^ℓ in the plaintext space, which is exactly the same as normal secret sharing computation. Therefore, AHE does not leak additional information as long as the security assumption of the AHE holds (hardness of integer factorization in the case of DGK). □

Using EOS for Differential Privacy. To use EOS for DP, each user encrypts one share (w.l.o.g., the r^{th} share) using the server's public key pk_s before uploading. In addition, we have the shufflers add fake reports. The full description of this protocol is given in Algorithm 5. There are three kinds of parties, users, shufflers, and the server. They all agree to use some method FO with the same parameter (e.g., ϵ, domain size, etc); the FO can be either GRR or SOLH, depending on the utility, as described in Section 9.3.3. All the communication is done through a secure channel. The users split their LDP reports into r shares, encrypt only the r-th shares using AHE, and send them to the shufflers. Each shuffler generate n_r shares for fake reports; only the r-th shuffler encrypt the shares with AHE. In this case, a malicious shuffler can draw its shares from a biased distribution; but those shares will then be "masked" by other honest shufflers' random shares and become uniformly random. By Corollary 9.5.1, the users' reports are protected from the shufflers;

Algorithm 5 PEOS

User i**:** Value v_i

$Y_i = \mathsf{FO}(v_i)$ ▷ FO can be GRR or SOLH

Split Y_i into r shares $\langle Y_{i,j} \rangle_{j \in [r]}$

for $j \in [r-1]$ **do**

  Send $Y_{i,j}$ to auxiliary server j

Send $c_{i,r} \leftarrow \mathrm{Enc}_{pk}(Y_{i,r})$ to auxiliary server r

Shuffler $j \in [r-1]$**:** Shares $\langle Y_{i,j} \rangle_{i \in [n]}$

for $k \in [n_r]$ **do** ▷ Generate shares of fake reports

  Sample $Y_{k,j}$ uniformly from output space of FO

Participate in EOS with $\langle Y_{i,j} \rangle_{i \in [n]}$ and $\langle Y_{k,j} \rangle_{k \in [n_r]}$ and send the shuffled result to the server

Shuffler r**:** Encrypted shares $\langle c_{i,r} \rangle_{i \in [n]}$

for $k \in [n_r]$ **do** ▷ Encrypted shares of fake reports

  Sample $Y_{k,r}$ uniformly from output space of FO

  $c_{k,r} \leftarrow \mathrm{Enc}_{pk}(Y_{k,r})$

Participate in EOS with $\langle c_{i,r} \rangle_{i \in [n]}$ and $\langle c_{k,r} \rangle_{k \in [n_r]}$ and send the shuffled result to the server

Server: Shares from auxiliary servers

Decrypt and aggregate the shares to recover Y

For any $v \in D$, estimate f_v using Y and Equation (9.5)

and the server cannot learn the permutation unless he can corrupt more than half of the auxiliary servers.

9.5.2 Privacy Analysis

Now we analyze the privacy guarantee of PEOS. Because of the usage EOS protocol, the server knows all the fake reports and each user's LDP report if it can corrupt more than $\lfloor r/2 \rfloor$ of the shufflers. And in this case, each user's privacy is only protected by ϵ_l-DP. On the other hand, as long as the server cannot corrupt more than $\lfloor r/2 \rfloor$ shufflers, the server cannot gain useful information.

In what follows, we assume the server cannot corrupt more than $\lfloor r/2 \rfloor$ shufflers and examine the privacy guarantee of PEOS. The focus is on how the privacy guarantees change after the addition of n_r fake reports. With these injected reports, what the server can observe

is the reports from both users and the shufflers. If the users collude, the server can subtract all other users' contribution and the privacy comes from the fake reports. The following corollaries give the precise privacy guarantee:

Corollary 9.5.2. *If* SOLH *is used and* SOLH *is ϵ_l-LDP, then* PEOS *is ϵ_c-DP against the server; and if other users collude with the server, the protocol is ϵ_s-DP, where*

$$\begin{aligned}\epsilon_s &= \sqrt{14\ln(2/\delta)\cdot\frac{d'}{n_r}}\\ \epsilon_c &= \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d'-1}+\frac{n_r}{d'}\right)}\end{aligned} \tag{9.6}$$

Proof: The proof is similar to the setting of with SOLH, but with n_r more random reports. More specifically, when other users collude, privacy is provided by the n_r random reports that are always random, and follow uniform distribution over $[d']$. Plugging the argument into Equation (9.2), these can be viewed as a random variable that follows Binomial distribution with $\mathcal{B}\left(n_r, \frac{1}{d'}\right)$. The rest of the proof follows from that for Theorem 9.3.2.

Similarly, for the privacy guarantee against the server, there are $n-1$ random reports from users, and n_r reports from the auxiliary server. Their effects can be viewed as one Binomial random variable: $\mathcal{B}\left(n-1, 1/(e^{\epsilon_l}+d'-1)\right)+\mathcal{B}\left(n_r, 1/d'\right) = \mathcal{B}\left(n-1+n_r, \frac{(n-1)/(e^{\epsilon_l}+d'-1)+n_r/d'}{n-1+n_r}\right)$. □

One can also use GRR in PEOS, and we have a similar theorem:

Corollary 9.5.3. *If* GRR *is used and* GRR *is ϵ_l-LDP, then* PEOS *is ϵ_c-DP against the server; and if other users collude with the server, the protocol is ϵ_s-DP, where*

$$\begin{aligned}\epsilon_s &= \sqrt{14\ln(2/\delta)\cdot\frac{d}{n_r}}\\ \epsilon_c &= \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d-1}+\frac{n_r}{d}\right)}\end{aligned}$$

The proof is similar to that for Corollary 9.5.2 and is thus omitted.

9.5.3 Utility Analysis

In Section 9.3.3, we analyze the accuracy performance of different methods under the basic shuffling setting. In this section, we further analyze the utility of these methods in PEOS. The difference mainly comes from the fact that n_r dummy reports are inserted, and the server runs a further step (i.e., Equation (9.5)) to post-process the results. In what follows, we first show that Equation (9.5) gives an unbiased estimation; based on that, we then provide a general form of estimation accuracy.

We first show f_v is an unbiased estimation of f_v, where $f_v = \frac{1}{n}\sum_{i\in[n]} \mathbf{1}_{v_i=v}$.

Lemma 5. *The server's estimation f_v from Equation (9.5) is an unbiased estimation of f_v, i.e.,*$\mathrm{E}\left[\tilde{f}_v\right] = f_v$.

Proof:

$$\mathrm{E}\left[f_v\right] = \mathrm{E}\left[\frac{n+n_r}{n}\tilde{f}_v - \frac{n_r}{n}\frac{1}{d}\right] = \frac{n+n_r}{n}\mathrm{E}\left[\tilde{f}_v\right] - \frac{n_r}{n}\frac{1}{d} \tag{9.7}$$

Here $\tilde{f}_v$ is the estimated frequency of value v given the $n+n_r$ reports; among them, n of them are from the true users, and n_r are from the randomly sampled values. For the n reports from users, nf_v of them have original value v; and for the n_r reports, in expectation, n_r/d of them have original value v. After perturbation, we have

$$\mathrm{E}\left[\tilde{f}_v\right] = \frac{nf_v + n_r/d}{n+n_r}$$

Putting it back to Equation (9.7), we have $\mathrm{E}\left[\tilde{f}_v\right] = f_v$. □

Given that, we prove the expected squared error of f_v:

$$\mathrm{Var}\left[f_v\right] = \mathrm{Var}\left[\frac{n+n_r}{n}\tilde{f}_v - \frac{n_r}{n}\frac{1}{d}\right] = \frac{(n+n_r)^2}{n^2}\mathrm{Var}\left[\tilde{f}_v\right]$$

Now plugging in the results of $\mathrm{Var}\left[\tilde{f}_v\right]$ from Section 9.3.3 (note that we use replace n with $n+n_r$ in the denominator as there are $n+n_r$ total reports), we obtain the specific variance of different methods after inserting n_r dummy reports.

Corollary 9.5.2 gives both ϵ_s and ϵ_c. For ϵ_s, d' is fixed given n_r and δ; but we can vary d' given ϵ_c. In particular, we can also derive the optimal value of d' following the similar to the analysis of Section 9.3.3 (after Proposition 9.3.5):

Given $\epsilon_c = \sqrt{14\ln(2/\delta)/\left(\frac{n-1}{e^{\epsilon_l}+d'-1}+\frac{n_r}{d'}\right)}$, we have

$$e^{\epsilon_l} + d' - 1 = \frac{n-1}{14\ln(2/\delta)/\epsilon_c^2 - n_r/d'}$$

We denote it as m, and (to simplify the notations) use a to represent $14\ln(2/\delta)/\epsilon_c^2$ and b to represent $n-1$. By the variance derived above, we have $\text{Var} = \frac{m^2}{(m-d)^2(d-1)}\frac{n+n_r}{n^2}$. Note that this formula is similar to the previous one in Section 9.3.3; but here m also depends on d'. Thus we need to further simplify Var:

$$\frac{(n+n_r)\left(\frac{b}{a-n_r/d'}\right)^2}{n^2\left(\frac{b}{a-n_r/d'}-d\right)^2(d'-1)} = \frac{(n+n_r)b^2}{n^2\left(b-(a-n_r/d)d'\right)^2(d'-1)} = \frac{(n+n_r)b^2}{n^2a^2\left(d'-(b+n_r)/a\right)^2(d'-1)}$$

To minimize Var, we want to maximize $\left(d'-(b+n_r)/a\right)^2(d'-1)$. By making its partial derivative to 0, we can obtain that when

$$d' = \frac{(b+n_r)/a+2}{3} = \frac{\epsilon_c^2(n-1-n_r)}{42\ln(2/\delta)} + \frac{2}{3}$$

the variance is minimized. Comparing to Equation (9.4), introducing n_r will reduce the optimal d'. We use the integer component of d' in the actual implementation.

9.5.4 Discussion and Guideline

PEOS strengthens the security aspect of the shuffler model from three perspectives: First, it provides better privacy guarantee when users collude with the server, which is a common assumption made in DP. Second, it makes the threat of the server colluding with the shufflers

more difficult. Third, it limits the ability of data poisoning of the shufflers. We discuss criteria for initiating PEOS.

Choosing Parameters. Given the desired privacy level $\epsilon_1, \epsilon_2, \epsilon_3$ against the three adversaries $\mathsf{Adv}, \mathsf{Adv}_u, \mathsf{Adv}_a$, respectively. Also given the domain size d, number of users n, and δ, we want to configure PEOS so that it provides $\epsilon_c \leq \epsilon_1$, $\epsilon_s \leq \epsilon_2$, and $\epsilon_l \leq \epsilon_3$.

Local perturbation is necessary to satisfy ϵ_3-DP against Adv_a. To achieve ϵ_2 when other users collude, noise from auxiliary servers are also necessary. Given that, to satisfy $\epsilon_c \leq \epsilon_1$, if we have to add more noise, we have two choices. That is, the natural way is to add noisy reports from the auxiliary server, but we can also lower ϵ_l at the same time. As we have the privacy and utility expressions, we can numerically search the optimal configuration of n_r and ϵ_l. Finally, given ϵ_l, we can choose to use either GRR or SOLH by comparing Theorem 9.3.2 and Theorem 9.3.3.

9.6 Evaluation

The purpose of the evaluation is two-fold. First, we want to measure the utility of SOLH, i.e., how much it improves over exsiting work. Second, we want to measure the communication and computation overhead of PEOS, to see whether the technique is applicable in practice.

As a highlight, our PEOS can make estimations that has absolute errors of $< 0.01\%$ in reasonable settings, improving orders of magnitude over existing work. The overhead is small and practical.

9.6.1 Experimental Setup

Datasets. We run experiments on three real datasets.

- IPUMS [34]: The US Census data for the year 1940. We sample 1% of users, and use the city attribute (N/A are discarded). This results in $n = 602325$ users and $d = 915$ cities.

- Kosarak [62]: A dataset of 1 million click streams on a Hungarian website that contains around one million users with 42178 possible values. For each stream, one item is randomly chosen.

- AOL [63]: The AOL dataset contains user queries on AOL website during the first three months in 2006. We assume each user reports one query (w.l.o.g., the first query), and limit them to be 6-byte long. This results a dataset of around 0.5 million queries including 0.12 million unique ones. It is used in the succinct histogram case study in Section 9.6.3.

Competitors. We compare the following methods:

- OLH: The local hashing method with the optimal d' in the LDP setting [5].

- Had: The Hadamard transform method used in [64]. It can be seen as OLH with $d' = 2$ (utility is worse than OLH); but compared to OLH, its server-side evaluation is faster.

- SH: The shuffler-based method for histogram estimation [51].

- AUE: Method from [57]. It first transforms each user's value using one-hot encoding. Then the values (0 or 1) in each location is incremented w/p $p = 1 - \frac{200}{\epsilon_c^2 n} \ln(4/\delta)$. Note that it is not an LDP protocol, and its communication cost is $O(d)$.

- RAP: The hashing-based idea described in Section 9.3.1. Its local side method is equivalent to RAPPOR [2]. Similar to AUE, it has large communication cost.

- RAP_R: Method from [59]. Similar to AUE and RAP, it transforms each user's value using one-hot encoding. The method works in the removal setting of DP. When converting to the replacement definition, it has the same utility as RAP.

- SOLH: The hashing-based idea introduced in Section 9.3.2.

- PEOS: We focus on the perspective of the computation and communication complexity in Section 9.6.4.

- SS: As a baseline, we also evaluate the complexity of the sequential shuffling method presented in 9.5.1; we call it SS.

Implementation. The prototype was implemented using Python 3.6 with fastecdsa 1.7.4, pycrypto 2.6.1, python-xxhash 1.3.0 and numpy 1.15.3 libraries. For SS, we generate a random AES key to encrypted the message using AES-128-CBC, and use the ElGamal encryption with elliptic curve secp256r1 to encrypt the AES key. For the AHE in PEOS, we use DGK [65] with 3072-bits ciphertext. All of the encryption used satisfy 128-bit security.

Metrics. We use *Mean Squared Error* (MSE) of the estimates as metrics. For each value v, we compute its estimated frequency $\tilde{f}_v$ and the ground truth f_v, and calculate their squared difference. Specifically, MSE $= \frac{1}{|D|}\sum_{v\in D}(f_v - \tilde{f}_v)^2$.

Methodology. For each dataset and each method, we repeat the experiment 100 times, with result mean and standard deviation reported. The standard deviation is typically very small, and barely noticeable in the figures. By default, we set $\delta = 10^{-9}$.

9.6.2 Frequency Estimation Comparison

We first show the utility performance of SOLH. We mainly compare it against other methods in the shuffler model, including SH, AUE, RAP, and RAP$_\mathsf{R}$. For comparison, we also evaluate several kinds of baselines, including LDP methods OLH and Had, centralized DP method Laplace mechanism (Lap) that represents the lower bound, and a method Base that always outputs a uniform distribution.

Figure 9.3 shows the utility comparison of the methods. We vary the overall privacy guarantee ϵ_c against the server from 0.1 to 1, and plot MSE. First of all, there is no privacy amplification for SH when ϵ_c is below a threshold. In particular, when $\epsilon_c < \sqrt{\frac{14\ln(2/\delta)d}{n-1}}$, $\epsilon_l = \epsilon_c$. We only show results on the IPUMS dataset because for the Kosarak dataset, d is too large so that SH cannot benefit from amplification. When there is no amplification, the utility of SH is poor, even worse than the random guess baseline method. Compared to SH, our improved SOLH method can always enjoy the privacy amplification advantage, and gets better utility result, especially when ϵ_c is small. The three unary-encoding-based methods AUE, RAP, and RAP$_\mathsf{R}$ are all performing similar to SOLH. But the communication cost of

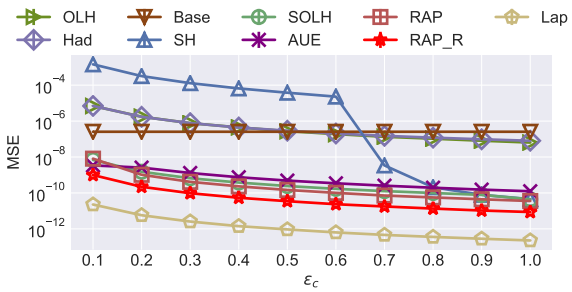


Figure 9.3. Results of MSE varying ϵ_c on the IPUMS dataset. Base always outputs $1/d$ for each estimation. Lap stands for Laplace mechanism for DP.

Table 9.2. Comparison of SOLH and RAP_R in Kosarak.

Metric	Method \ ϵ_c	0.2	0.4	0.6	0.8
d'	SOLH	45	177	397	705
Utility	SOLH	5.27e-8	1.30e-8	5.76e-9	3.24e-9
	RAP_R ($d' = 10$)	1.31e-7	1.17e-7	1.14e-7	1.13e-7
	RAP_R ($d' = 100$)	1.73e-7	1.55e-8	1.22e-8	1.22e-8
	RAP_R ($d' = 1000$)	1.02e-4	2.60e-5	4.02e-8	3.66e-9
	RAP_R	7.82e-9	1.92e-9	8.53e-10	4.78e-10

them are higher. The best-performing method is RAP_R; but it works in the removal-LDP setting. Because of this, its performance with ϵ_c is equivalent to RAP with $2\epsilon_c$.

Moving to the LDP methods, OLH and Had perform very similar (because in these settings, OLH mostly chooses $d' = 2$ or 3, which makes it almost the same as Had), and are around 3 orders of magnitude worse than the shuffler-based methods. For the central DP methods, we observe Lap outperforms the shuffler-based methods by around 2 orders of magnitude.

In Table 9.2, we list the value of d' of SOLH and the utility of SOLH and RAP_R for some ϵ_c values. We also fix d' in SOLH and show how sub-optimal choice of d' makes SOLH less accurate. The original domain d is more than 40 thousand, thus introducing a large communication cost compared to SOLH (5KB vs 8B). The computation cost for the users is low for both methods; but for the server, estimating frequency with SOLH requires evaluating hash functions. We note that as this takes place on server, some computational

cost is tolerable, especially the hashing evaluation nowadays is efficient. For example, our machine can evaluate the hash function 1 million times within 0.1 second on a single thread.

9.6.3 Succinct Histograms

In this section, we apply shuffle model to the problem of succinct histogram (e.g., [8], [15]) as a case study. The succinct histogram problem still outputs the frequency estimation; but different from the ordinary frequency or histogram estimation problem, which we focused on in the last section, it handles the additional challenge of a much larger domain (e.g., domain size greater than 2^{32}). To deal with this challenge, [15] proposes TreeHist. It assumes the domain to be composed of fixed-length binary strings and constructs a binary prefix tree. The root of the tree denotes the empty string. Each node has two children that append the parent string by 0 and 1. For example, the children of root are two prefixes $0*$ and $1*$, and the grand children of root are $00*$, $01*, 10*$, and $11*$. The leaf nodes represent all possible strings in the domain.

To find the frequent strings, the algorithm traverses the tree in a breadth-first-search style: It starts from the root and checks whether the prefixes at its children are frequent enough. If a prefix is frequent, its children will be checked in the next round. For each round of checking, an LDP mechanism (such as those listed in Chapter 3) is used. Note that the mechanism can group all nodes in the same layer into a new domain (smaller than the original domain because many nodes will be infrequent and ignored). Each user will check which prefix matches the private value, and report it (or a dummy value if there is no match). In this section, to demonstrate the utility gain of the shuffler model, we use the methods SH, SOLH, AUE, and RAP as the frequency estimator (i.e., the framework of TreeHist stays the same; but the frequency estimator is changed).

In what follows, we empirically compare them to demonstrate the applicability and benefit of the shuffler model. Following the setting of [15], we consider the AOL dataset assuming each user's value is 48 bits. We run TreeHist in 6 rounds, each for 8 bits (1 character). We set the goal to identify the the top 32 strings, and in each intermediate round, we identify the top 32 prefixes. In the LDP setting, TreeHist divides the users into 6 groups, as that

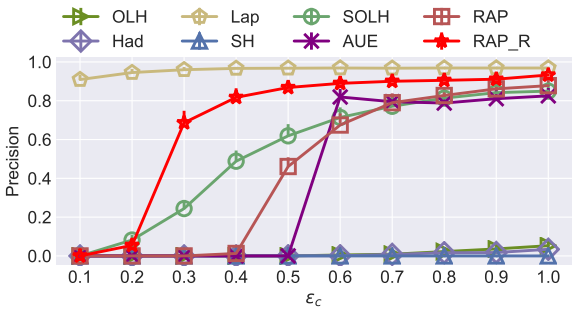


Figure 9.4. Comparison on the succinct histogram problem. The target is to identify the top 32 most frequent values.

gives better results. In the shuffler case, a better approach is to avoid grouping users, but rather dividing ϵ_c and δ_c by 6 for each round.

Figure 9.4 shows the results. We can observe that the except SH, the other shuffler-based methods outperforms the LDP TreeHist (OLH and Had) . In addition to the capability of reducing communication cost, another advantage of SOLH we observe here is that SOLH enables non-interactive execution of TreeHist (note that this is also one reason why the original TreeHist algorithm uses the local hashing idea). In particular, the users can encode all their prefixes and report together. The server, after obtaining some frequent prefix, can directly test the potential strings in the next round. On the other hand, using the unary-encoding-based methods, users cannot directly upload all their prefixes, because the size of a report can be up to 2^{48} bits. Instead, the server has to indicate which prefixes are frequent to the users and then request the users to upload.

9.6.4 Performance Evaluation

We evaluate the computational and communication costs of SS and PEOS, focusing on the overhead introduced by the encryption and shuffling. We run the experiments on servers running Linux kernel version 5.0 with Intel Xeon Silver 4108 CPU @ 1.80GHz and 128GB memory. We assume there are $r = 3$ and $r = 7$ shufflers. The results are listed in Table 9.3. As both methods scales with $n + n_r$, we fix n to be 1 million and ignore n_r.

Table 9.3. Computation and communication overhead of SS and PEOS for each user, each shuffler, and the server. We assume $n = 10^6$ and $r = 3$ or 7.

Metric \ Method	SS		PEOS	
	$r = 3$	$r = 7$	$r = 3$	$r = 7$
User comp. (ms)	0.24	0.49	1.6	1.6
User comm. (Byte)	416	800	400	432
Aux. comp. (s)	49	50	0.2	0.7
Aux. comm. (MB)	224	416	429.8	3293.3
Server comp. (s)	49	49	65	65
Server comm. (MB)	128	128	392	408

Note that we the results are only for SOLH with report size fixed at 64 bits. If we use RAP in this case, the communication cost will increase proportional to the size of the domain d (by $d/64$).

User Overhead. Overall, the user-side computation and communication overhead are small for both methods. The computation only involves sampling, secret-sharing, and r times of encryption operations. All of them are fast. Note that in SS, as onion encryption is used, its overhead is larger and grows linearly with respect to r. The communication cost for each user is also very limited.

Shuffler Overhead. For each shuffler in SS, the computation cost lies in n decryptions (for one layer), sampling n_u random reports (with necessary encryption), and then shuffling. Note that the decryptions is done in parallel. In this implementation, we use 32 threads for demonstration. With more resources, the processing time can be shortened.

In SS, an ElGamal ciphertext is a tuple $\langle P, C\rangle$, P is a point in the secp256r1 curve and thus can be represented by 256×2 bits; and C is a number in $\{0,1\}^{256}$. That is, we need 96 bytes to store the encrypted AES key for each layer. For SOLH, we let each user randomly select an 4-byte seed as the random hash function. After padding, each message is $32 + 96(r + 1)$ bytes, where r is the number of layers used for shufflers. One additional layer is used for the server. Given $n = 1$ million users and r shufflers, there will be on average $\frac{1}{r} \times n \times \sum_{k=1}^{r}(32 + 96(k + 1)) = 672$ MB data sent to the three shufflers.

PEOS is made up of $\binom{r}{\lfloor r/2 \rfloor + 1}$ rounds of sorting. Since a well-implemented sorting on 1 million elements takes only several milliseconds, the computation cost of shuffling is minor for the shufflers. In addition, our protocol require each shuffler do $\binom{r}{\lfloor r/2 \rfloor + 1} \cdot n/r$ homomorphic additions during shuffling. As Table 9.3 indicates, all of these cryptographic operations are efficient. The cost is no more than one second with $n = 1$ million reports.

According to the analysis of oblivious shuffle from [55], each shuffler's communication cost is $O(2^r \sqrt{r} n)$. In addition, our protocol sends n encrypted shares each round, which introduces another communication cost of $O(2^r n/\sqrt{r})$ by similar analysis (multiplied with a larger constant factor because of the 3072-bit DGK ciphertexts). In experiments with 1 million users and 3 shufflers, each shuffler needs to send 430 MB. In a more expensive case with 7 shufflers, it becomes 3.3 GB. While the communication cost is higher than that of SS, we note that the cost is tolerable in our setting, as the data collection does not happen frequently.

Server Overhead. For SS, the server computation overhead is similar to that of the shufflers, as they all decrypt one layer. The server's communication cost (measured by amount of data received) is lower though, as there is only one layer of encryption on the data.

In PEOS, the server needs to collect data from all r shufflers. As only one share is encrypted by DGK, the communication overhead is mostly composed of that part and grows slowly with r. The computation overhead is also dominated by decrypting the DGK ciphertexts.

9.7 Related Work

Privacy Amplification by Shuffling. The shuffling idea was originally proposed in Prochlo [56]. Later the formal proof was given in [49]–[51]. Parallel to our work, [57], [66] propose mechanisms to improve utility in this model. They both rely on the privacy blanket idea [51]. More recently, [59] considered an intriguing removal-based LDP definition and

work in the shuffler model. Besides estimating histograms, the problem of estimating the sum of numerical values are also extensively investigated [58], [67], [68].

Crypto-aided Differential Privacy. Different from using shufflers, researchers also proposed methods that utilize cryptography to provide differential privacy guarantees, including [69]–[71]. One notable highlight is [52], which proposes Cryptϵ. In this approach, users encrypt their values using homomorphic encryption, and send them to the auxiliary party via a secure channel. The auxiliary server tallies the ciphertext and adds random noise in a way that satisfies centralized DP, and sends the result to the server. The server decrypts the aggregated ciphertext. More recently, researchers in [72] introduce several security features including verification and malice detection. This line of work does not require LDP protection, thus differs from our approach. Moreover, to handle the histogram estimation when $|D|$ is larger, the communication overhead is larger than that of ours.

Relaxed Definitions. Rather than introducing the shuffler, another direction to boost the utility of LDP is to relax its *semantic meaning*. In particular, Wang et al. propose to relax the definition by taking into account the distance between the true value and the perturbed value [73]. More formally, given the true value, with high probability, it will be perturbed to a nearby value (with some pre-defined distance function); and with low probability, it will be changed to a value that is far apart. A similar definition is proposed in [74], [75]. Both usages are similar to the geo-indistinguishability notion in the centralized setting [76]. In [77], the authors consider the setting where some answers are sensitive while some not (there is also a DP counterpart called One-sided DP [78]). The work [79] is a more general definition that allows different values to have different privcay level. Our work applied to the standard LDP definition, and we conjecture that these definitions can also benefit from introducing a shuffler without much effort.

There also exist relaxed models that seem incompatible with the shuffler model, i.e., [80] considers the inferring probability as the adversary's power; and [35] utilizes the linkage between each user's sensitive and public attributes.

Distributed DP. In the distributed setting of DP, each data owner (or proxy) has access to a (disjoint) subset of users. For example, each patient's information is possessed by a

hospital. The DP noise is added at the level of the intermediate data owners (e.g., [81]). A special case (two-party computation) is also considered [82], [83]. [84] studies the limitation of two-party DP. In [85], a distributed noise generation protocol was proposed to prevent some party from adding malicious noise. The protocol is then improved by [86]. [87] lays the theoretical foundation of the relationship among several kinds of computational DP definitions.

We consider a different setting where the data are held by each individual users, and there are two parties that collaboratively compute some aggregation information about the users.

DP by Trusted Hardware. In this approach, a trusted hardware (e.g., SGX) is utilized to collect data, tally the data, and add the noise within the protected hardware. The result is then sent to the analyst. Google propose Prochlo [56] that uses SGX. Note that the trusted hardware can be run by the server. Thus [88] and [89] designed oblivious DP algorithms to overcome the threat of side information (memory access pattern may be related to the underlying data). These proposals assume the trusted hardware is safe to use. However, using trusted hardware has potential risks (e.g., [90]). This paper considers the setting without trusted hardware.

REFERENCES

[1] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006.

[2] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: randomized aggregatable privacy-preserving ordinal response," in *CCS*, 2014.

[3] Apple Differential Privacy Team, *Learning with privacy at scale, available at https://machinelearning.apple.com/research/learning-with-privacy-at-scale*, 2017.

[4] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *NIPS*, 2017.

[5] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *USENIX Security*, 2017.

[6] T. Wang, J. Blocki, N. Li, and S. Jha, "Optimizing locally differentially private protocols," *arXiv preprint arXiv:1705.04421*, 2017.

[7] *Source code of rappor in chromium*, https://cs.chromium.org/chromium/src/components/rappor/public/rappor_parameters.h.

[8] R. Bassily and A. D. Smith, "Local, private, efficient protocols for succinct histograms," in *STOC*, 2015.

[9] *Scikit-learn*, http://scikit-learn.org/.

[10] *Frequent itemset mining dataset repository*, http://fimi.ua.ac.be/data/.

[11] T. Wang, N. Li, and S. Jha, "Locally differentially private heavy hitter identification," *IEEE TDSC*, 2019.

[12] T. Wang, N. Li, and S. Jha, "Locally differentially private heavy hitter identification," *CoRR*, vol. abs/1708.06674, 2017.

[13] T.-H. H. Chan, E. Shi, and D. Song, "Private and continual release of statistics," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 3, pp. 1–24, 2011.

[14] G. C. Fanti, V. Pihur, and Ú. Erlingsson, "Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries," *PoPETs*, vol. 2016, no. 3, 2016.

[15] R. Bassily, K. Nissim, U. Stemmer, and A. G. Thakurta, "Practical locally private heavy hitters," in *NIPS*, 2017.

[16] N. Wang, X. Xiao, Y. Yang, T. D. Hoang, H. Shin, J. Shin, and G. Yu, "Privtrie: Effective frequent term discovery under local differential privacy," in *ICDE*, 2018.

[17] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, "Boosting the accuracy of differentially private histograms through consistency," *PVLDB*, vol. 3, no. 1, 2010.

[18] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits, "Blender: Enabling local search with a hybrid differential privacy model," in *USENIX Security*, 2017, pp. 747–764.

[19] *Quantcast top sites*, https://www.quantcast.com/top-sites/, 2016.

[20] T. Wang, N. Li, and S. Jha, "Locally differentially private frequent itemset mining," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 127–143.

[21] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *CCS*, 2016.

[22] N. Li, W. Qardaji, and D. Su, "On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ACM, 2012, pp. 32–33.

[23] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *arXiv preprint arXiv:1709.02753*, 2017.

[24] Z. Zhang, T. Wang, N. Li, S. He, and J. Chen, "Calm: Consistent adaptive local marginal for marginal release under local differential privacy," in *CCS*, 2018.

[25] G. Cormode, T. Kulkarni, and D. Srivastava, "Marginal release under local differential privacy," in *SIGMOD*, 2018.

[26] X. Ren, C. Yu, W. Yu, S. Yang, X. Yang, J. A. McCann, and P. S. Yu, "Lopub: High-dimensional crowdsourced data publication with local differential privacy," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 9, 2018.

[27] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar, "Privacy, accuracy, and consistency too: A holistic solution to contingency table release," in *PODS*, 2007.

[28] W. Qardaji, W. Yang, and N. Li, "Priview: Practical differentially private release of marginal contingency tables," in *SIGMOD*, 2014.

[29] T. T. Nguyên, X. Xiao, Y. Yang, S. C. Hui, H. Shin, and J. Shin, "Collecting and analyzing data from smart device users with local differential privacy," *arXiv:1606.05053*, 2016.

[30] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Traces and emergence of nonlinear programming*, Springer, 2014, pp. 247–258.

[31] W. Karush, "Minima of functions of several variables with inequalities as side constraints," *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*, 1939.

[32] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2001, pp. 401–406.

[33] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[34] S. Ruggles, S. Flood, R. Goeken, J. Grover, E. Meyer, J. Pacas, and M. Sobek, *Integrated public use microdata series: Version 5.0 [machine-readable database]*, 2020.

[35] T. Wang, B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha, "Answering multi-dimensional analytical queries under local differential privacy," in *SIGMOD*, 2019.

[36] W. H. Qardaji, W. Yang, and N. Li, "Understanding hierarchical methods for differentially private histograms," *PVLDB*, vol. 6, no. 14, 2013.

[37] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.," in *KDD*, vol. 96, 1996.

[38] X. Xiao, G. Wang, and J. Gehrke, "Differential privacy via wavelet transforms," in *ICDE*, 2010.

[39] T. Wang, Z. Li, N. Li, M. Lopuhaä-Zwakenberg, and B. Skoric, "Consistent and accurate frequency oracles under local differential privacy," in *NDSS*, 2020.

[40] G. Cormode, T. Kulkarni, and D. Srivastava, "Answering range queries under local differential privacy," *PVLDB*, 2019.

[41] P. Kairouz, K. Bonawitz, and D. Ramage, "Discrete distribution estimation under local privacy," in *ICML*, 2016.

[42] R. Bassily, "Linear queries estimation with local differential privacy," in *AISTATS*, 2019.

[43] J. Jia and N. Z. Gong, "Calibrate: Frequency estimation and heavy hitter identification with local differential privacy via incorporating prior knowledge," 2019.

[44] B. Ding, M. Winslett, J. Han, and Z. Li, "Differentially private data cubes: Optimizing noise sources and consistency," in *SIGMOD*, 2011.

[45] J. Lee, Y. Wang, and D. Kifer, "Maximum likelihood postprocessing for differential privacy under consistency constraints," in *KDD*, 2015.

[46] J. Blasiok, M. Bun, A. Nikolov, and T. Steinke, "Towards instance-optimal private query release," in *SODA*, 2019.

[47] Q. Ye, H. Hu, X. Meng, and H. Zheng, "Privkv: Key-value data collection with local differential privacy," in *SP*, 2019.

[48] T. Wang, B. Ding, M. Xu, Z. Huang, C. Hong, J. Zhou, N. Li, and S. Jha, "Improving utility and security of the shuffler based differential privacy," *VLDB*, 2020.

[49] A. Cheu, A. D. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *EUROCRYPT*, 2019.

[50] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta, "Amplification by shuffling: From local to central differential privacy via anonymity," in *SODA*, 2019, pp. 2468–2479.

[51] B. Balle, J. Bell, A. Gascón, and K. Nissim, "The privacy blanket of the shuffle model," in *Annual International Cryptology Conference*, Springer, 2019, pp. 638–667.

[52] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha, "Crypte: Crypto-assisted differential privacy on untrusted servers," *SIGMOD*, 2020.

[53] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 223–238.

[54] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*, Springer, 2008, pp. 192–206.

[55] S. Laur, J. Willemson, and B. Zhang, "Round-efficient oblivious database manipulation," in *International Conference on Information Security*, Springer, 2011, pp. 262–277.

[56] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *SOSP*, ACM, 2017.

[57] V. Balcer and A. Cheu, "Separating local & shuffled differential privacy via histograms," in *1st Conference on Information-Theoretic Cryptography (ITC 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[58] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, R. Pagh, and A. Velingker, "Pure differentially private summation from anonymous messages," *arXiv:2002.01919*, 2020.

[59] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, S. Song, K. Talwar, and A. Thakurta, "Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation," *arXiv preprint arXiv:2001.03618*, 2020.

[60] I. Damgard, M. Geisler, and M. Kroigard, "Homomorphic encryption and secure comparison," *Int. J. Appl. Cryptol.*, vol. 1, no. 1, pp. 22–31, Feb. 2008.

[61] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.)," *IEEE Transactions on Information Theory*, 1978.

[62] *Frequent itemset mining dataset repository*, Available at http://fimi.ua.ac.be/data/.

[63] *Web search query log downloads*, Available at http://www.radiounderground.net/aol-data/.

[64] J. Acharya, Z. Sun, and H. Zhang, "Hadamard response: Estimating distributions privately, eciently, and with little communication," in *AISTATS*, 2019.

[65] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Australasian Conference on Information Security and Privacy*, Springer, 2007, pp. 416–430.

[66] B. Ghazi, N. Golowich, R. Kumar, R. Pagh, and A. Velingker, "On the power of multiple anonymous messages," *arXiv:1908.11358*, 2019.

[67] B. Ghazi, R. Pagh, and A. Velingker, "Scalable and differentially private distributed aggregation in the shuffled model," *arXiv preprint arXiv:1906.08320*, 2019.

[68] B. Balle, J. Bell, A. Gascon, and K. Nissim, "Private summation in the multi-message shuffle model," in *CCS*, 2020.

[69] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux, "Unlynx: A decentralized system for privacy-conscious data sharing," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 232–250, 2017.

[70] T. Elahi, G. Danezis, and I. Goldberg, "Privex: Private collection of traffic statistics for anonymous communication networks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1068–1079.

[71] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient private statistics with succinct sketches," in *NDSS*, 2016.

[72] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, "Honeycrisp: Large-scale differentially private aggregation without a trusted core," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ACM, 2019, pp. 196–210.

[73] S. Wang, Y. Nie, P. Wang, H. Xu, W. Yang, and L. Huang, "Local private ordinal data distribution estimation," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, IEEE, 2017, pp. 1–9.

[74] M. E. Gursoy, A. Tamersoy, S. Truex, W. Wei, and L. Liu, "Secure and utility-aware data collection with condensed local differential privacy," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[75] X. Gu, M. Li, Y. Cao, and L. Xiong, "Supporting both range queries and frequency estimation with local differential privacy," in *2019 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2019, pp. 124–132.

[76] M. Andrés, N. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *20th ACM Conference on Computer and Communications Security*, ACM, 2013, pp. 901–914.

[77] T. Murakami and Y. Kawamoto, "Utility-optimized local differential privacy mechanisms for distribution estimation," in *28th USENIX Security Symposium*, 2019.

[78] S. Doudalis, I. Kotsogiannis, S. Haney, A. Machanavajjhala, and S. Mehrotra, "One-sided differential privacy," *arXiv preprint arXiv:1712.05888*, 2017.

[79] X. Gu, M. Li, L. Xiong, and Y. Cao, "Providing input-discriminative protection for local differential privacy," in *ICDE*, 2020.

[80] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, "Protection against reconstruction and its applications in private federated learning," *arXiv:1812.00984*, 2018.

[81] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, vol. 3, 2017.

[82] X. He, A. Machanavajjhala, C. Flynn, and D. Srivastava, "Composing differential privacy and secure computation: A case study on scaling private record linkage," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1389–1406.

[83] F.-Y. Rao, J. Cao, E. Bertino, and M. Kantarcioglu, "Hybrid private record linkage: Separating differentially private synopses from matching records," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 3, p. 15, 2019.

[84] A. McGregor, I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan, "The limits of two-party differential privacy," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, IEEE, 2010, pp. 81–90.

[85] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 486–503.

[86] J. Champion, J. Ullman, *et al.*, "Securely sampling biased coins with applications to differential privacy," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2019, pp. 603–614.

[87] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Annual International Cryptology Conference*, Springer, 2009, pp. 126–142.

[88] T. H. Chan, K.-M. Chung, B. M. Maggs, and E. Shi, "Foundations of differentially oblivious algorithms," in *SODA*, SIAM, 2019.

[89] J. Allen, B. Ding, J. Kulkarni, H. Nori, O. Ohrimenko, and S. Yekhanin, "An algorithmic framework for differentially private data analysis on trusted processors," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 635–13 646.

[90] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi, "The guard's dilemma: Efficient code-reuse attacks against intel sgx," in *27th USENIX Security Symposium*, 2018.

VITA

Tianhao Wang was born and raised in Zibo, China. He move to Shanghai and attended High School Affiliated to Shanghai Jiao Tong University. He graduated from Fudan University with a Bachelor of Engineering in Software Engineering in 2015, under the supervision of Dr. Yunlei Zhao. Tianhao entered Purdue University in the Fall of 2015, and worked under the supervision of Dr. Ninghui Li in the Department of Computer Science. Tianhao's graduate research was in the area of differentially private data publishing. He received his Ph.D. in computer science in Spring of 2021.