

Locally Differentially Private Heavy Hitter Identification

Tianhao Wang

Purdue University

tianhaowang@purdue.edu

Ninghui Li

Purdue University

ninghui@cs.purdue.edu

Somesh Jha

University of Wisconsin-Madison

jha@cs.wisc.edu

Abstract—The notion of Local Differential Privacy (LDP) enables users to answer sensitive questions while preserving their privacy. The basic LDP frequent oracle protocol enables the aggregator to estimate the frequency of any value. But when the domain of input values is large, finding the most frequent values, also known as the heavy hitters, by estimating the frequencies of all possible values, is computationally infeasible. In this paper, we propose an LDP protocol for identifying heavy hitters. In our proposed protocol, which we call Prefix Extending Method (PEM), users are divided into groups, with each group reporting a prefix of her value. We analyze how to choose optimal parameters for the protocol and identify two design principles for designing LDP protocols with high utility. Experiments on both synthetic and real-world datasets demonstrate the advantage of our proposed protocol.

I. INTRODUCTION

In recent years, differential privacy [11], [12] has been increasingly accepted as the *de facto* standard for data privacy in the research community [3], [13], [20], [22], [27]. Recently, techniques for satisfying differential privacy (DP) in the local setting, which we call LDP, have been deployed. Such techniques enable gathering of statistics while preserving privacy of every user, without relying on trust in a single data curator. For example, researchers from Google developed RAPPOR [14], [17], which is included as part of Chrome. It enables Google to collect users' answers to questions such as the default homepage of the browser, the default search engine, and so on, to understand the unwanted or malicious hijacking of user settings. Apple [1] also uses similar methods to help with predictions of spelling and other things, but the details of

the algorithm are not public yet. Samsung proposed a similar system [26] which enables collection of not only categorical answers (e.g., screen resolution) but also numerical answers (e.g., time of usage, battery volume), although it is not clear whether this has been deployed by Samsung.

In the LDP setting, each user possesses an input value $v \in D$, and the aggregator wants to learn the distribution of the input values among all users. Existing research [6], [14], [30] has developed frequency oracle protocols, where the aggregator can estimate the frequency of any chosen value $v \in D$. When the size of D is small, such frequency oracle protocols can be used to efficiently reconstruct a noisy approximation of the input distribution. When the size of D is so large that issuing an oracle query for each value in it is computationally infeasible, one needs an additional protocol that first identifies a set of candidate frequent values. Two protocols for doing this exist [6], [17].

In this paper, we propose the Prefix Extending Method (PEM), which is conceptually very simple, and yet is able to provide much better accuracy than existing protocols, and the advantage is more pronounced as the size of D gets larger. The basic idea of PEM is to gradually identifying longer and longer frequent prefixes. For example, if we view D as consisting of length- m binary strings, we divide the users into g groups, where users in each group report a prefix of a certain length. Users in the $j + 1$ 'th group report prefixes of length η longer than the j 'th group, and the g 'th group report the whole string. Thus the population is divided into g groups of roughly the same size. The aggregator uses reports from the first group to finds C_1 , the set of frequent prefixes, and then uses reports from the second group to find C_2 , considering candidates that have prefixes in C_1 . The aggregator iterates this process until finding the set of frequent values.

An important parameter in this process is the segment length η . Larger η would mean higher computational cost. In terms of utility, larger η means fewer groups and more users in each group, which improves utility. However, larger η means more candidates to consider in each step, which leads

to lower accuracy. We conduct an utility analysis to study the interactions of these two effects. The utility analysis enables us to draw to a conclusion that the first effect dominates the second, and thus larger η results in better utility. Thus the choice of η depends on limitation on the computational resources. Because of the complexity of the problem, we have to make several simplifying approximations in the analysis. To validate the analysis, we accompany each step of the analysis with empirical experiment to show that conclusions drawn from the analysis match empirical ones.

With the analysis, we are able to identify two design principles. The first is, when asking multiple questions, it is better to partition the users into groups, and having each group answer one question, as opposed to having each user answer all the questions splitting the privacy budget. The second is, one should reduce the number of groups as much as possible when designing LDP protocols, as larger group size is a key in achieving accuracy. Some existing protocols violate these principles, and can be improved by following them. We expect these principles to guide the design of LDP protocols for other problems.

Finally, we demonstrate the effectiveness of PEM by conducting experiments with both synthetic and real-world datasets. Result shows that PEM greatly outperforms existing solutions.

To summarize, we make the following contributions:

- We provide new solutions for the privacy-preserving heavy hitter problem. The protocol is then analyzed and optimized.
- We identify two principles that can guide the design of protocols for other LDP problems.
- We demonstrate the effectiveness of our solution using real-world and synthetic datasets.

Roadmap. In Section II, we present LDP and describe existing mechanisms. We then go over the problem definition and existing solutions in Section III. Section IV presents our proposed method and analysis. The analysis is validated in Section V. Experiment results are given in VI. Finally we discuss related work in Section VII and conclude in Section VIII.

II. BACKGROUND

We consider a setting where there are many *users* and one *aggregator*. Each user possesses an input value $v \in D$, and the aggregator wants to learn (and use) the distribution of the input values among all users, in a way that protects the privacy of individual users.

In the standard (or centralized) setting, each user sends v to the aggregator, which obtains a histogram for the distribution, and can add noises to the histogram to satisfy differential privacy, so that each individual user's input has a limited impact on the output. In this setting, the aggregator sees the raw input from all users and is trusted to handle these private data correctly.

A. Differential Privacy in Local Setting

In the local (or distributed) setting, we want to remove the need to trust the aggregator. To achieve this, each user perturbs the input value v using an algorithm π and sends $\pi(v)$ to the aggregator. The formal privacy requirement is that the algorithm $\pi(\cdot)$ satisfies local differential privacy, defined as follows:

Definition 1 (Local Differential Privacy): An algorithm π satisfies ϵ -local differential privacy (ϵ -LDP), where $\epsilon \geq 0$, if and only if for any input $v_1, v_2 \in D$, we have

$$\forall T \subseteq Range(\pi) : \Pr[\pi(v_1) \in T] \leq e^\epsilon \Pr[\pi(v_2) \in T],$$

where $Range(\pi)$ denotes the set of all possible outputs of the algorithm π .

For an algorithm $\pi(\cdot)$ to satisfy ϵ -LDP, it must be randomized. 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.

B. Frequency Oracles

A protocol that satisfies LDP is specified by two algorithms: π , which is used by each user to perturb her input value, and Γ , which takes as input the reports from all users, and outputs the desired information. A basic protocol under LDP is to estimate the frequency of any given value $v \in D$. In which a protocol, Γ outputs an oracle that can be queried for the frequency of each value. We thus call such a protocol a frequency oracle protocol.

We assume that there are n users, and user j 's value is $v^j \in D$, and the domain size is $|D| = d$.

1) Generalized Randomized Response (GRR): One frequency oracle protocol generalizes the *randomized response* technique [31]. In this protocol, $\pi_{GRR}(v)$ outputs the value v with probability $p = \frac{e^\epsilon}{e^\epsilon + d - 1}$, and each value $v' \neq v$ with probability $\frac{1-p}{d-1} = \frac{1}{e^\epsilon + d - 1} = \frac{p}{e^\epsilon}$. In the special case where the value is one bit, i.e., when $d = 2$, $\pi_{GRR}(v)$ keeps the bit unchanged with probability $\frac{e^\epsilon}{e^\epsilon + 1}$ and flips it with probability $\frac{1}{e^\epsilon + 1}$.

The frequency oracle outputted by Γ_{GRR} in this protocol works as follows. To estimate the frequency of v , it counts

how many times v is reported and obtains I_v , and then outputs $\frac{I_v - nq}{p - q}$. That is, the frequency estimate is a linear transformation of the noisy count I_v , in order to account for the effect of randomized response. In [30], it is shown that this is an unbiased estimation of the true count, and the variance for this estimation is

$$\frac{d - 2 + e^\epsilon}{(e^\epsilon - 1)^2} \cdot n. \quad (1)$$

The accuracy of this protocol deteriorates fast when the domain size d increases. The larger d is, the lower the probability that a value is preserved. This is reflected in the fact that the variance of is linear in d . For example, when $\epsilon = \ln 49$, with $d = 2^{16}$, we have $p = \frac{49}{65584} \approx 0.00075$, and variance $\frac{65583}{2304} \approx 28.5n$

More sophisticated frequency estimators have been studied before [6], [14], [30]. In [30], several such protocols are analyzed, optimized, and compared against each other, and it was found that when d is large, the Optimized Local Hashing (OLH) protocol provides the best accuracy while maintaining a low communication cost. In this paper, we use the OLH protocol as a primitive and describe it below.

2) *Optimized Local Hashing (OLH)* [30]: The Optimized Local Hashing (OLH) protocol deals with a large domain size d by first using a hash function to map an input value into a smaller domain of size d' , typically $d' \ll d$, and then applies randomized response to the value in the smaller domain. In this protocol, both the hashing step and the randomization step result in information loss. The choice of the parameter d' is a tradeoff between losing information during the hashing step and losing information during the randomization step. In [30], it is found that the optimal choice of d' is $e^\epsilon + 1$.

In OLH, $\pi_{OLH}(v) = \langle H, \pi_{GRR}(H(v)) \rangle$, where H is randomly chosen from a family of hash functions that hash each value in D to $\{1 \dots d'\}$, where $d' = \lceil e^\epsilon + 1 \rceil$, and π_{GRR} is the perturb algorithm used in generalized randomized response, with probability $p = \frac{e^\epsilon}{e^\epsilon + d' - 1}$.

The frequency oracle outputted by Γ_{OLH} in this protocol works as follows. Let $\langle H^j, y^j \rangle$ be the report from the j 'th user. For each value $v \in D$, the oracle first computes $I_v = |\{j \mid H^j(v) = y^j\}|$. That is, I_v is the number of reports that “supports” that the input is v . The oracle then outputs

$$\frac{I_v - n/d'}{p - 1/d'}. \quad (2)$$

The variance of this estimation is

$$\frac{4e^\epsilon}{(e^\epsilon - 1)^2} \cdot n. \quad (3)$$

Compared with (1), the factor $d - 2 + e^\epsilon$ is replaced by $4e^\epsilon$. This suggests that for smaller d , one is better off with GRR; but for large d , OLH is better and has a variance that does not depend on d .

We point out that using OLH, 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 group size. One may notice that the above frequency oracles under LDP all have estimation variance that is linear in n , which means that the standard deviation of the estimations is linear in \sqrt{n} . This is shared by all protocols under LDP, and is a fundamental accuracy cost one has to pay in order to achieve LDP [8]. What this means, however, is that LDP protocols can be useful only when the group size n is large, and LDP protocols are meaningful only for the frequent values.

We now use some concrete numbers to make these points clear. For example, to recover a value that is possessed by 0.1% of the population, we have the true count being $0.001n$. Assuming we choose ϵ such that $e^\epsilon = 10$, then using OLH the standard deviation is $\sqrt{\frac{40}{81}n} \approx 0.7\sqrt{n}$. If we desire that the true count is at least 3 times the standard deviation, then we require $0.001n \geq 3 \times 0.7\sqrt{n}$, or $n \geq 4,410,000$. This suggest that with about 4.5 million users, we can recover meaningful frequencies for values that appear in at least 0.1% of the population. Quadrupling the population size would enables us to reduce this 0.1% sensitivity threshold by half. While theoretically there are up to 1000 values with frequencies 0.1% or higher, in most distributions there are likely no more than a few dozen of such values, because the most frequent values will appear with frequencies far higher than 0.1% and the total frequencies of infrequent values can also be substantial.

III. PROBLEM DEFINITION AND EXISTING METHODS

Recall that the aggregator wants to know the distribution of the frequent values. When the data domain D is relatively small, having a frequency oracle protocol 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 paper we focus on the problem of identifying frequent values under the 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.

A. Problem Definition

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. Also, note that when the population size n and the privacy budget ϵ is set, the number of threshold above which one can estimate frequencies accurately is more or less fixed. We use the top- k version of definition.

Definition 2 (Top- k Heavy Hitter): Given a multi-set $\{v^1, v^2 \dots, v^n\} \in D^n$. An element $x \in D$ is a top- k heavy hitter if its frequency $f_x = \frac{|\{j|j \in [n] \wedge v^j = x\}|}{n}$ 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 goal is to identify a set of candidates from the domain D , such that it is computationally feasible to query the frequency oracle.

B. Strawman Method

To better understand the protocols proposed in [6], [17], we start by describing a strawman method for identifying a smaller set of candidates for frequent values. An intuitive method is to divide and conquer. Specifically, a length- m value is divided into g equal-size segments, each of length $s = m/g$. For example, when $g = 8, m = 128$, each segment has $s = 16$ bits. Borrowing Python list syntax, we use the notation $v[i : j]$ to denote the segment of v starting at the i 'th bit and stopping at (and including) the $j - 1$ 'th bit. Thus $v[0 : m]$ represents the complete v .

In the strawman protocol, each user randomly chooses a segment to report. More specifically, the user randomly chooses $1 \leq \alpha \leq g$, and reports

$$\langle \pi(v), \alpha, \pi(v[(\alpha - 1)s : \alpha s]) \rangle,$$

where π can be any perturb function, although it is natural to use OLH. That is, the whole population is divided (by their own random choices) into g groups, each reporting on one segment.

The aggregator first queries the frequency of each length- s binary string in each of the g segments, issuing a total of $2^s \times g$ oracle queries, and identifying the frequent patterns in each segment. Let C_1, \dots, C_g denote the frequent patterns for the g segments. The candidate set C is the Cartesian product of C_i , i.e., $C = C_1 \times C_2 \times \dots \times C_g$, 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. Finally, the aggregator

queries frequencies of these candidates, using the full string reports $\pi(v)$.

The main shortcoming of this method is that, if we identify k candidates from each of C_1, \dots, C_g , the candidate set C has size k^g . When m is large, g is not very small, and the candidate set C is still too large to be enumerated. The protocols proposed in [6], [17] can be viewed as taking two different approaches in further improving this method.

C. The Segment Pairs Method (SPM) [17]

The approach taken by Google's team for the RAPPOR system improves upon the above strawman protocol by having each user report a pair of two randomly chosen segments, instead of reporting only one segment. We call this the Segment Pair Method (SPM).

In SPM, the length- m value is divided into g segments of length $s = m/g$. 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

$$\langle \pi(v), \alpha, \beta, \pi(v[(\alpha - 1)s : \alpha s]), \pi(v[(\beta - 1)s : \beta s]) \rangle.$$

That is, 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. Then, it queries, for each pair $1 \leq i, j \leq g$ of segments, the frequency for the values in $C_i \times C_j$ 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 $g = 8$, each pair has only about $\frac{n}{28}$ users. And when $g = 16$, each pair has only about $\frac{n}{120}$ users.

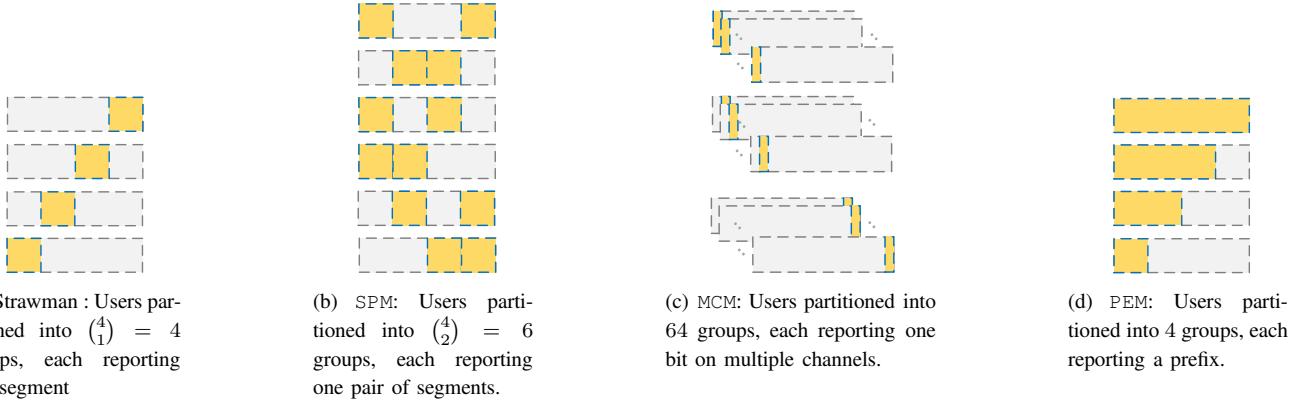


Fig. 1. Illustration of how candidate frequent values are generated by different methods. Assuming that each value has 64 bits, often divided into 4 segments of 16-bits each. The first three methods require in addition the ability to estimate the frequency of any value in the whole domain. In SPM and MCM, this is done by having each user report both the segments in the figure and the whole value, dividing the privacy budget. We show that it is better to divide the population to have another group of users who report only the whole value. In PEM, the last group serves the purpose.

D. The Multiple Channel Method (MCM) [6]

Bassily and Smith proposed an approach which we call Multiple Channel Method (MCM) [6]. Our description of MCM below simplifies that in [6], and is equivalent to it. This approach can be viewed as improving upon the strawman approach by using a technique to separate the values into multiple channels so that with high probability each channel has at most one frequent value, and then identifying 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 $\{1 \dots h\}$. 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 ℓ such that $0 \leq \ell < m$ and reports:

$$\langle \pi(v), \ell, b_1, b_2, \dots, b_h \rangle$$

The privacy budget ϵ is divided into two parts $\epsilon_1 + \epsilon_2 = \epsilon$. Sending the value v uses ϵ_1 ; b_1, b_2, \dots, b_h 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 $v[\ell]$, 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 [6], 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.

IV. PROPOSED SOLUTION

In both SPM and MCM, 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. MCM 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 have these segments *overlapping*. In our proposed method, which we call Prefix Extending Method (PEM), users in each group report a prefix of her value. Figure 1 illustrates the differences between the four methods we have discussed. The main advantage of PEM over other methods is that when m is long, one needs to divide the population only into g groups.

A. Prefix Extending Method (PEM)

The PEM method is parameterized by two parameters γ and η , which are positive integers. A user is randomly assigned into

one of g groups, where $g = \lceil \frac{m-\gamma}{\eta} \rceil$. The assignment can be made by the aggregator, or having each user selecting a group at random. Users in the i 'th group where $1 \leq i \leq g$ report

$$\langle i, \pi(v[0 : \gamma + i\eta]) \rangle.$$

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

Note that here we assume that we have no domain knowledge about the underlying values and thus represent the values as bit strings and divide it into equal-length segments. The basic idea of PEM, where one iteratively find portions of the whole values that are frequent, can be applied in other contexts. In a given application, one can take advantage of domain-specific knowledge to define segments differently. For example, one can use the domain knowledge to eliminate candidates that are impossible. If the values have internal structures such as one component can have values of different lengths, one can also extend that component in one step and test values of different lengths. In this paper, we focus on the binary string setting.

B. Protocol Analysis

The PEM protocol has two parameters γ and η . Typically, γ should be slightly larger than η , to make the candidate set size roughly the same in each step. The choice of η , however, is very important. Larger η would mean higher computational cost. Furthermore, by having a large η , there will be fewer groups, and thus more users in each group, making the estimation in each step more accurate; on the other hand, there will be more values to consider in each step, thus the probability a non-heavy hitter is identified is increased. We now analyze the utility to optimize the choice of η .

1) Metric: To compare utility when using different η values, we use the following utility measurements.

F-measure (F1). Define v_j as the j -th most frequent value. The ground truth for top k values is $C_T = \{v_1, v_2, \dots, v_k\}$. Denote the k values identified by the protocol using 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 [23] which is the harmonic mean of precision and recall, i.e.,

$$F1 = \frac{2}{1/P + 1/R} = \frac{2PR}{P+R}$$

$$\text{where } P = \frac{|C_T \cap C_g|}{|C_g|}, R = \frac{|C_T \cap C_g|}{|C_T|}$$

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 [19]:

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

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.

Unified Utility Score. We express the utility scores as the weighted average of the identification probability of each heavy hitter, that is,

$$\sum_{j=1}^k \left(w_j \cdot \prod_{i=1}^g Piden_j[i] \right), \quad (4)$$

where $Piden_j[i]$ is the identification probability for the j -th most frequent value v_j in step i , that is, $v_j[0 : \gamma + i\eta] \in C_i$. We will elaborate $Piden_j[i]$ later. The overall identification probability is the product of that of each phase.

Different metrics can be expressed by different weights. In the F-measure, $w_j = \frac{1}{k}$, and for NCR, where the higher ranked value receives greater weight, $w_j = \frac{k+1-j}{\sum_{i=1}^k k+1-i}$.

In our analysis below, we assume that the identification of the heavy hitters are mutually-independent. Technically this is not true. If one value has been identified as a heavy hitter, the probability that another one is identified will be slightly lower, since we are identifying k heavy hitters. However, when k is not very small, this effect is small and can be ignored for our purpose. We will empirically verify the correctness of this approximation.

2) *Assumptions and Constraints:* We first simplify PEM by initiate some parameters. Recall that there are two main parameters γ, η in PEM. Users are partitioned into $g = \lceil \frac{m-\gamma}{\eta} \rceil$ groups. We assume that the number of users in all groups are the same; thus, the number of users in the i 'th group is $n[i] = n/g$. We fix the size of output in each stage to be $|C_i| = k$. We further fix $\gamma = \lceil \log_2 k \rceil$. Thus the aggregator makes $|D_i| = 2^{\gamma+\eta} = k \cdot 2^\eta$ queries to the frequency oracle in each step, and the only parameter left for us to choose is η .

To calculate the actual utility scores, we have to make some assumptions of the dataset distribution. This is because the significance (frequency) of the heavy hitters will affect the utility measure. For example, in an almost uniformly distributed dataset, it is hard to find out the most frequent k values, since the frequency differences are very small. We also limit the maximum allowed number of frequency oracle queries.

3) *Approximate Identification Probability:* We now calculate $\text{Piden}_j[i]$, the probability v_j 's prefix is identified in step i , i.e., $v_j[0 : \gamma + i\eta] \in C_i$.

We first show the estimation of a value is a random variable. Assume the true frequency of v_j is f_j . $n[i] = n/g$ users are randomly assigned to report the first $\gamma + i\eta$ bits of their private value, and each of them possesses the value v_j with probability f_j . By (2), estimation of v_j is only determined by the “support”, I_j , it receives. Since we only care about the relative ranks of the estimations, we focus on I_j and use it as estimation of v_j . For each user, if he has value v_j (with probability f_j), his report will “support” v_j (reports $H^j(v_j)$ in terms of OLH) with probability p ; otherwise his report will “support” v_j with probability $q = 1/d'$. Therefore, I_j can be seen as the summation of $n[i]$ binomial variables, whose probability of being 1 is $p_j = p \cdot f_j + q \cdot (1 - f_j)$. In most cases (as long as $n[i]$ is large), we can approximate I_j using normal distribution with mean $\mu_j[i] = n[i] \cdot p_j$ and variance $\sigma_j^2[i] = n[i] \cdot p_j \cdot (1 - p_j)$.

We then calculate the probability $v_j[0 : \gamma + i\eta]$'s estimation is ranked on top k . Since I_j is a normal random variable, we know the probability a value is estimated above any threshold value T , that is, $\Pr[I_j > T]$. For all the non-heavy hitters, summing this up gives us the expected number of values that are estimated above T . If this expected number is less than k , then $\Pr[I_j > T]$ is the probability its estimation is ranked on top k . We use $T_k[i]$ to denote this threshold value. For efficiency, we assume that among all the values to be tested, D_i , all the $N[i] = |D_i| - k = k(2^\eta - 1)$ non-heavy values have zero frequencies (this is especially safe when $N[i]$ is large). Therefore, $T_k[i]$ can be calculated by the inverse of cumulative

density function:

$$T_k[i] = -\Phi^{-1}\left(\frac{k}{N[i]}\right) \cdot \sigma_0[i] + \mu_0[i]$$

where $\sigma_0^2[i] = n[i] \cdot q \cdot (1 - q)$, $\mu_0[i] = n[i] \cdot q$ denotes the variance and mean of these $N[i]$ zero-mean values.

Finally, we account for the effect of other heavy hitters. We assume the estimations of the top k values are always sorted, thus for the j -th value to be estimated top k , there are $k - j + 1$ slots, since the top $j - 1$ values are already ranked higher. Formally,

$$\begin{aligned} \text{Piden}_j[i] &= 1 - \Phi\left(\frac{T_{k-j}[i] - \mu_j[i]}{\sigma_j[i]}\right) = \Phi\left(\frac{\mu_j[i] - T_{k-j}[i]}{\sigma_j[i]}\right) \\ &= \Phi\left(\frac{\mu_j[i] + \Phi^{-1}\left(\frac{k-j}{N[i]}\right) \cdot \sigma_0[i] - \mu_0[i]}{\sigma_j[i]}\right) \end{aligned} \quad (5)$$

C. Instantiate PEM

With (5) to instantiate (4), we can now calculate the utility scores when using different values for η . Before the actual numerical computation, we need to assume the dataset distribution. For instance, we can assume that the users' value form a zipf's distribution, i.e., $f_j \propto \frac{1}{j}$. We also need to limit the number of total queries to the frequency estimator, e.g., 2^{20} queries.

The optimization inputs are: k as the number of desired heavy hitters, m as the domain size in bits, n as the number of users, and ϵ as the privacy budget. With all parameters available, η is instantiated with different values and the corresponding utility scores are calculated. The configuration that gives best utility score will be used.

In the actual optimization, we can make small changes to more parameters, namely, γ , and in different steps i , the number of users $n[i]$, and the candidate size $|C_i|$. It is also possible to use different η for each step, denoted by $\eta[i]$. It turns out that slightly change in these parameters does not affect the final result much. What affect utility the most is the number of groups one needs to divide the users into. When η is very small, the overall utility will deteriorate a lot.

As a result, in most of the system settings, the optimal configuration is $\gamma = \log_2 k$, and for all i , $|C_i| = k$, $\eta[i] = \eta$, $n[i] \approx n/g$, where η is the maximal integer such that the total number of queries $2^{\gamma+\eta} \cdot g$ ($g = \lceil \frac{m-\gamma}{\eta} \rceil$) is less than the limit. In some extreme cases, e.g., when k is very big, $|C_i|$ is smaller than k , suggesting that when k is too large, since it is impossible to accurately recover k heavy hitters, one should simply try to find fewer. On the other hand, when k is small, $|C_i|$ can be large, so that the probability prefixes of the heavy hitters are included in C_i are increased.

D. Observations and Design Principles

By the optimization and the supporting analysis, we are able to answer the questions raised in the beginning of this section. Specifically, we make some statements that can help guide the design of protocols for not only the heavy hitter problem but also other LDP problems.

First of all, PEM assigns users to different groups, and let each user report a prefix. It is possible to design PEM such that requires each user answer all prefixes. Since all prefixes from the same value are obviously related, correlation information can be extracted. By (5), we observe the following:

Proposition 1: By partitioning the users into groups and letting each group answer a separate question, the identification probability will be higher than having each user split privacy budget to answer all the questions.

We provide a proof to support our choice in the appendix.

We note that a similar observation has been made in [30], based on comparing variances of frequency estimation. We can generalize these observations into the following principle for designing effective LDP protocols.

Principle 1: In LDP setting, one should divide the user population, instead of dividing the privacy budget.

We note that in the centralized DP setting, these are generally equivalent in terms of utility.

From numerical computation of optimal parameters, we make the following observations.

Observation 1: Within the query limit, greatest identification probability is achieved with the least number of groups.

This implies that, when multiple groups are necessary, η should be as big as possible (within query limit). We distill this as a second principle for designing LDP protocols.

Principle 2: When designing LDP protocols, one should minimize the number of groups one has to divide the user population into, as a large group size is a key in obtaining accurate answers.

A similar principle applies in the centralized setting as well. By reducing the number of steps that need privacy budget, one can often achieve better results. In PEM, in order to increase the final utility, it is inclined to allocate more users to the final group, or minimize work by reduce η in the final phase. Somewhat counter-intuitively, a more balanced allocation will be better.

Observation 2: If $\eta[i] = m/g$, best result is achieved when $n[i] = n/g$. Similarly, if $n[i] = n/g$, best result is achieved when $\eta[i] = m/g$.

The previous observation is made when one of $n[i]$ and $\eta[i]$ is balanced (i.e., $n[i] = n/g$ or $\eta[i] = m/g$). When either is unbalanced (perhaps due to e.g., the leftover bits in the final phase), theoretical optimal allocation of the other is unknown. These observations cannot be proven easily. We provide empirical support in the next section (specifically, Figure 3).

V. VALIDATION OF ANALYSIS

In this section, we run empirical experiment to validate the analysis. That is, we want to show the analysis matches empirical results (mainly (4) and (5)). Moreover, we verify the design principles we made as observations in Section IV-D.

We generate synthetic datasets follows zipf's distribution (i.e., the j -th most frequent value has frequency f_j proportional to $\frac{1}{j^{1.5}}$). The values of the heavy hitters are random. We then evaluate PEM on the synthetic datasets with reasonable parameters. All experiment runs 100 times. For each setting, we use points with error bar (if any) for empirical results, and a thin line with the same color for analytical results. The validation is carried out in the following three stages.

A. Identification Probability.

We first verify the correctness of the identification probability (i.e., (5)). We generate $n = 100000$ data points each with $m = 64$ bits. Users report with privacy budget $\epsilon = 1$.

Note that with $m = 64$ bits or more, it is not feasible to estimate the whole domain in a single round. For the purpose of demonstration, we instantiate PEM with $\gamma = 5, \eta = 10, |C_i| = 32$.

In Figure 2, we show the identification probability of values with different frequencies. We observe that, first of all, the analysis matches the empirical result pretty well. Moreover, when the other parameters remain the same, the more users (larger n), the shorter the domain length (smaller m), or the larger the privacy budget (larger ϵ), the better the overall result. This trend also matches the analysis.

B. Utility Scores under Different Configurations.

Having verified the analytical identification probability (5) for each single value, we now verify the analytical utility score (4), which is the linear combination of the identification probabilities for each heavy hitter.

We generate $n = 100000$ data points each represented by $m = 16$ bits, and use $\epsilon = 1$. Besides the zipfs distribution, we also test on another similar distribution, ‘zipfs(-20)’. The difference is that in zipfs(-20), the most frequent 20 values are dropped. As a result, the first few heavy hitters are not

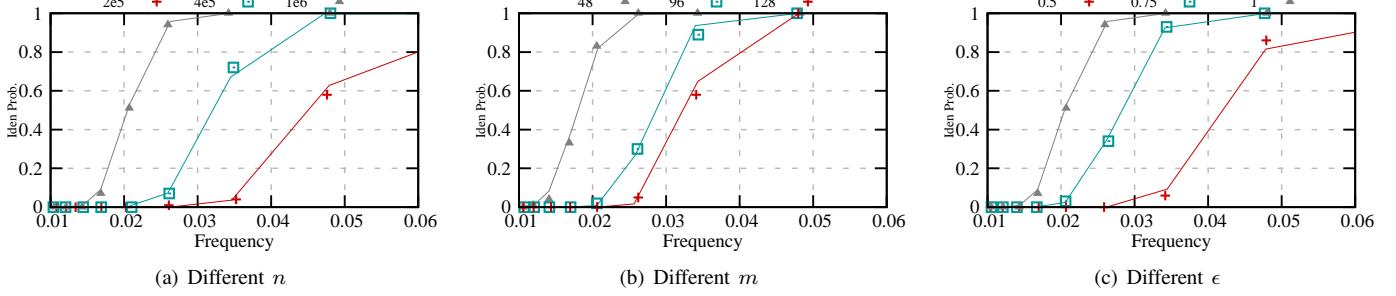


Fig. 2. Overall identification probabilities for values with different frequencies. Dots represent empirical results, and lines shows analytical results. We fix $n = 1000000$, $\epsilon = 1$, $m = 64$ as the default setting. In each sub-figure, we vary one of them while keeping the others.

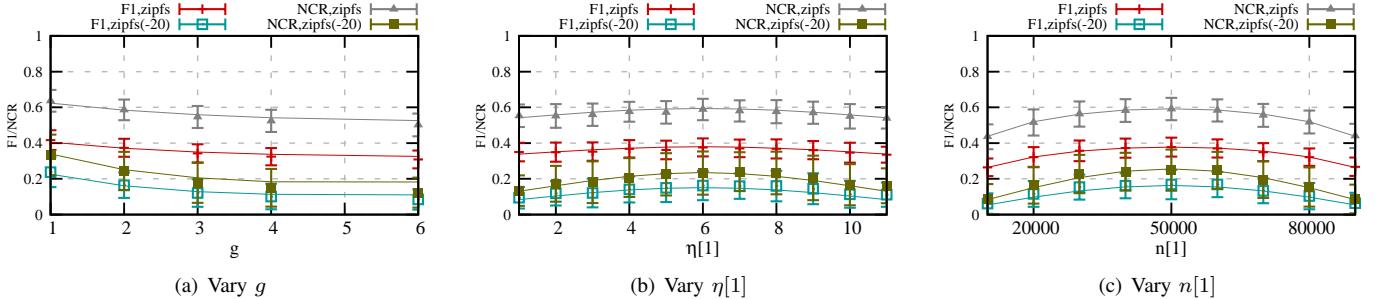


Fig. 3. Utility score from different configurations. We fix $k = 16$, and use the default configuration of $n = 100000$, $m = 16$, $\epsilon = 1$. In each sub-figure, we vary r , $s[1]$, $n[1]$, and plot F1, NCR for two distributions.

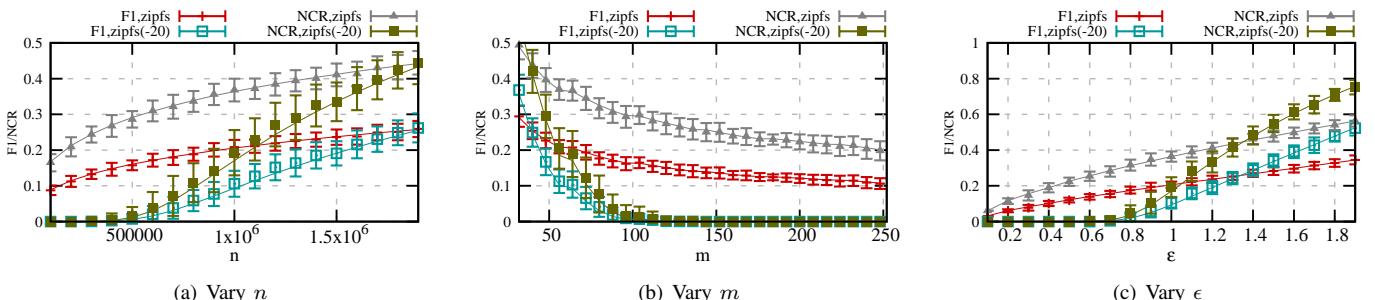


Fig. 4. Utility score from the optimal configuration. We fix $k = 32$, and use the default configuration of $n = 1000000$, $m = 64$, $\epsilon = 1$. In each sub-figure, we vary n , m , ϵ , and plot F1, NCR for two distributions.

that significantly frequent, but the following ones occurs more frequently.

The domain of this dataset is made smaller mainly for the purpose of making it clear when we compare different configurations of PEM. We plot F1 and NCR scores of the top $k = 16$ heavy hitters in Figure 3. It can be seen that all points (empirical results) lie on the line (analytical results), verifying that the analysis is accurate under different configurations.

Specifically, in Figure 3(a), we vary the number of rounds from 1 to 6, each outputting 16 candidates. As a result, with more rounds of test, the overall result becomes worse. This also validates principle (Observation 1) to have as few rounds as possible.

To support Observation 2, we fix $|C_i| = k = 16$, $\gamma = 4$

and $g = 2$, we run two sets of experiment. In the first one, we assign half of the users to each group, and vary segment size of the first round. It can be seen from Figure 3(b) that when $\eta[1] = 6$ (both rounds analyzes a domain of 2^{10} values), the overall result is optimal. Moreover, the result is symmetric on $\eta[1] = 6$ (since $\gamma = 4$, the effective number of bits to examine is 12). This is also the same as what we expect. In the second experiment, we fix $\eta = 6$ and try different user allocations , i.e., $n[1]$. As shown in Figure 3(c), when each of the two rounds receives half of the users, the overall result is maximized. Similar to the result for segment size, the result is symmetric on $n[1] = 50000$.

C. Optimal Utility Scores under Different Scenarios.

Now we verify the correctness of the optimal configuration under different scenarios. We use the default setting of $n = 1000000$, $m = 64$, $\epsilon = 1$, and plot F1 and NCR values of optimal PEM, varying any of n , m , and ϵ . The configuration is optimized taking F1 as the goal.

From Figure 4, we can first confirm analysis still matches empirical results well. When the setting is not “favorable” (ϵ is small, n is small, or m is large), zipfs distribution has better results, while zipfs(-20) gives similar or better result on the other extreme. The reason is that, when m is large and ϵ, n are not sufficiently big, the noise in PEM is large. In zipfs distribution, the first several heavy hitters are more significant, and therefore, the overall utility score is better. While on the other hand, it is possible to recover more heavy hitters. The following several heavy hitters in zipfs(-20) distribution, which are more frequent, contribute more to the overall result.

VI. 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.

A. Evaluation Setup

Each experiment is run 10 times, and the average and standard deviation are reported.

Datasets. 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 [4] 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.

1. Frequent URL. In SPM [17], 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 mimic the distribution by collecting a similar dataset from Quantcast [2]. 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, resulting a dataset containing 1.2 million data points, and 27 thousand unique urls.

This also motivates a real-world application, where the analyst can find out the most popular website. In a previous report of RAPPOR [14], the system collects homepage urls from users, by testing on a known list of websites. We focus on the scenario where the dictionary is unavailable or inaccurate.

2. Query Trends. The AOL dataset contains user queries on AOL website during the first three months in 2006. Similar to the settings of [4], we assume each user reports one query (w.l.o.g., the first query). The queries are limited to be 6 bytes long. This results a dataset of around 0.5 million queries including 0.2 million unique ones.

Many real-world application such as keyword trends or hot tags can be derived from this example. In these scenarios, the heavy hitters change frequently, such that the dictionary from history may not be reliable.

3. 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 [14].

Competitors. We consider the following algorithms: PEM, MCM, and SPM. In order to optimize PEM, we assume a zipf’s distribution, limit the number of queries to the frequency estimator to 2^{20} , and take F1 as the goal.

Both SPM and MCM were designed to find heavy hitters based on threshold, but PEM works for top k heavy hitters. For a fair comparison, we improve MCM and SPM in step 1 and 2, and change them from threshold based algorithms to top k based in step 3. Note that PEM can also be changed to work for threshold. The corresponding results are shown in Section VI-B3.

1. Replace LDP primitive. Existing methods use non-optimal LDP primitives, but they can be changed. Specifically, SPM use RAPPOR [14] as the internal LDP primitive, and MCM uses BLH. We replace RAPPOR and BLH with OLH to improve their efficiency.

2. 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 Observation 1, we make the segment length as long as possible, under the frequency oracle query limit.

MCM 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.

3. Replace Threshold Test. Existing methods require internal test and filtering based on a threshold. Specifically,

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.

In SPM, moreover, 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.

B. Detailed Results

1) *Effect of ϵ :* We show F1 and NCR results of different methods varying ϵ in Figure 5. 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, MCM and SPM can only identify two.

2) *Effect of k :* Figure 6 gives F1 and NCR results of different methods varying k . Similarly, we can see that PEM outperforms MCM and SPM. 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 $k = 30$. 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.

3) *Comparison of Threshold Version:* Both SPM and MCM use internal threshold test to find heavy hitters. 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 7. The results are similar in other datasets.

As can be seen from Figure 7(a), when we fix $\theta = 0.023$ (0.023 is around frequency of the 16-the most frequent value in the dataset), PEM performs better than SPM and MCM. This advantage is most profound when $\epsilon = 2$, where PEM achieves much better than existing methods. The effects of fixing ϵ and varying θ are also demonstrated in Figure 7(b) and 7(c).

4) *Effect of Partitioning Users:* We further improve existing algorithms according to Proposition 1. Namely, instead of split privacy budget, we allocate 10% of users for the final testing. The result shown in Figure 8(a) demonstrates the advantage of partitioning users. Especially, when $\epsilon = 1.2$, the original MCM 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.

5) *Effect of η :* In Figure 8(b), we demonstrate the effect of segment size, i.e., η , in PEM. We fix $\eta = 2, 4, 6, 8, 10$ and plot the results. It is clear that when η increases, the overall utility is better. When $\epsilon = 0.9$, we see the F1 score is 0.4 when $\eta = 2$, and 0.8 when $\eta = 10$. Note that there should be a limit on how large η can be, that is, η is limited by the number of queries the aggregator can make.

6) *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 average squared error as the metric, that is,

$$\text{Var} = \frac{1}{|C_T \cap C_g|} \sum_{v \in C_T \cap C_g} (n_v - \tilde{n}_v)^2,$$

where n_v is the true count of v and \tilde{n}_v is its estimation by the protocol. Note that we only account heavy hitters that are successfully identified by the protocol, i.e., $v \in C_T \cap C_g$.

Figure 9 shows comparison of estimation variance for different methods. Observe that the MCM method has smaller variance than SPM, because the final testing step of MCM uses half of the ϵ , while that of SPM uses one third. As a comparison, PEM uses only the last group, which is one sixth of users, and achieves similar estimation accuracy. Note that this also complies with (3), the estimation variance of OLH.

7) *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 8(c), under different assumptions, the results are very similar.

VII. RELATED WORK

Differential privacy has been the *de facto* notion protecting privacy. In the centralized settings, many DP algorithms have been proposed (see [13] for a theoretical treatment and [22] in a more practical perspective). Recently, Uber has deployed a system enforcing DP during SQL queries [20], Google also proposed several works that combine DP with machine

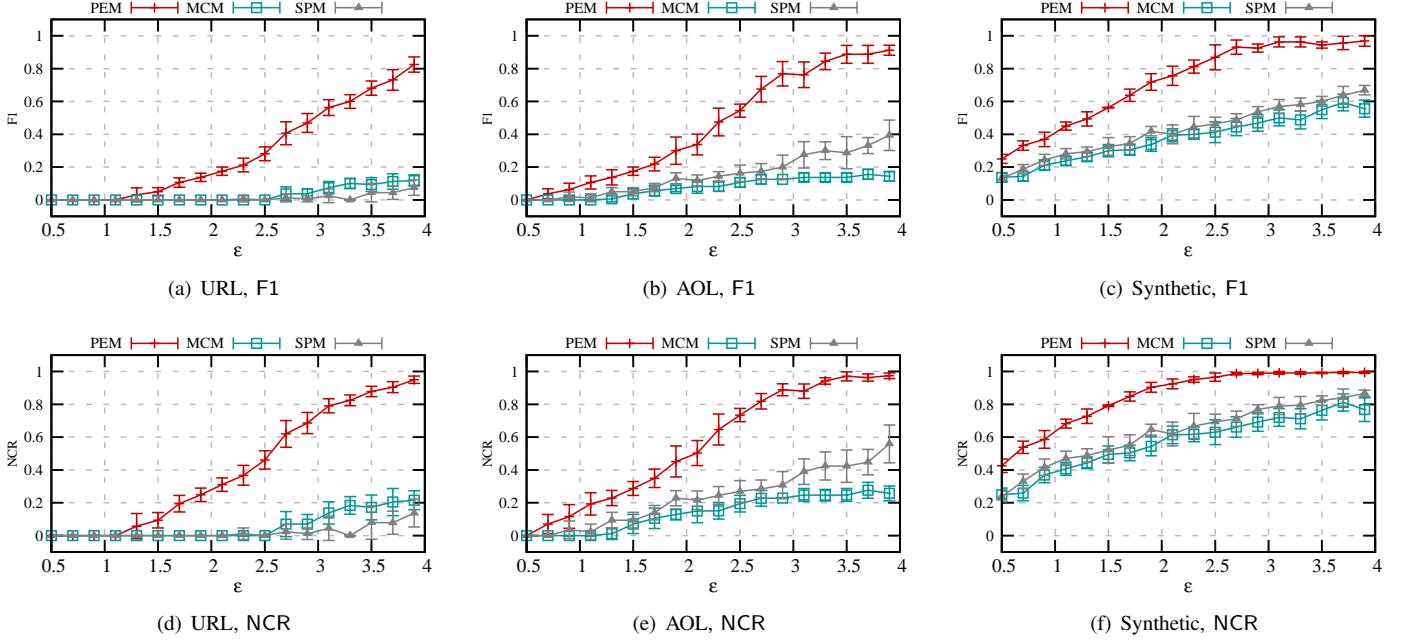


Fig. 5. Evaluation of the datasets, vary ϵ while fixing $k = 16$.

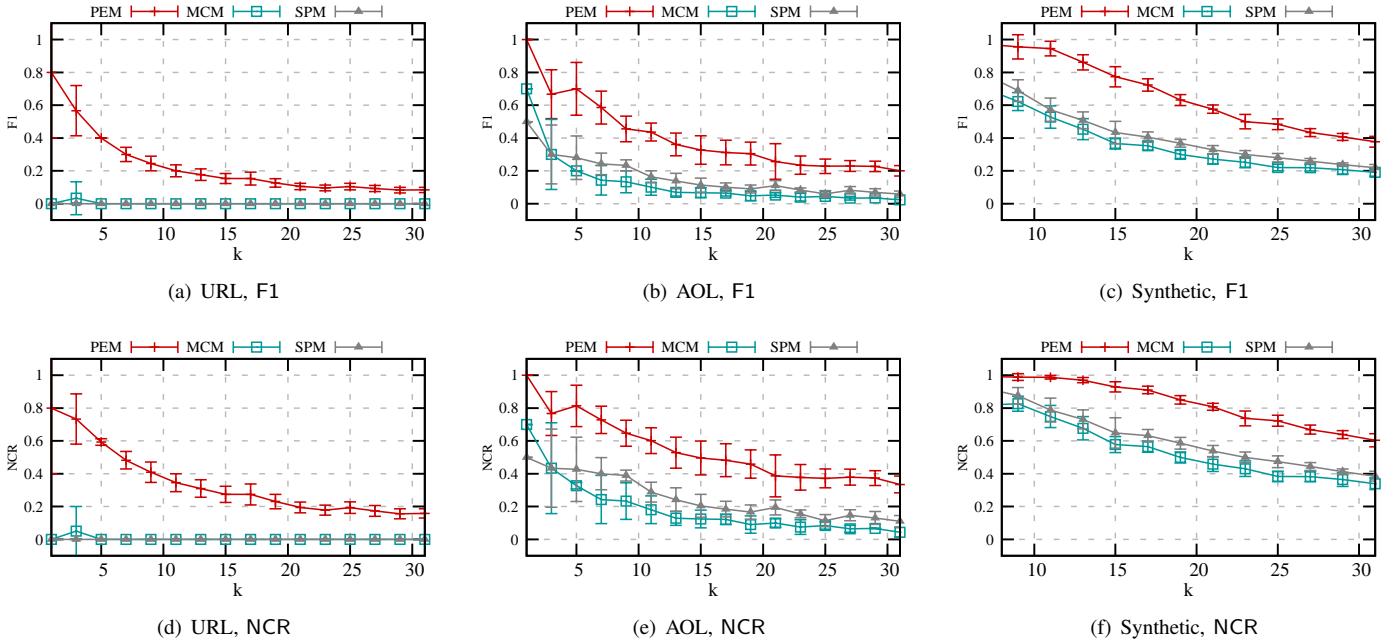


Fig. 6. Evaluation of the datasets, varying k while fixing $\epsilon = 2$.

learning [3], [27]. In the local setting, we have also seen real world deployment: Google deployed RAPPOR [14] as an extension within Chrome, and Apple [1] also uses similar methods to help with predictions of spelling and other things.

Of all the problems, the basic tools in LDP are mechanisms to estimate frequencies of values. Wang et al. compare different mechanisms using estimation variance [30]. They conclude that when the domain size is small, the Generalized Random Response provides best utility, and Optimal Local Hash

(OLH)/Optimal Unary Encoding (OUE) [30] when the domain is large. There also exists other mechanisms with higher variance: RAPPOR by Erlingsson et al. [14] and Random Matrix Projection (BLH) by Bassily and Smith [6]. These protocols use ideas from earlier work [10], [25]. Kairouz et al. [21] prove the optimal mechanisms are extreme.

The heavy hitter problem is to identify frequent values when the domain of possible values is very large, so that it is infeasible to obtain estimations for all values to identify

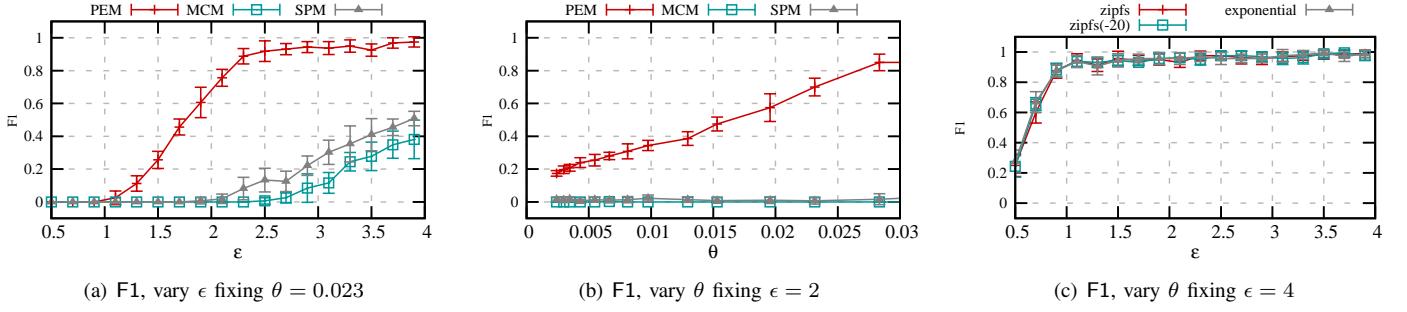


Fig. 7. Evaluation of the synthetic datasets, vary one of ϵ and θ while fixing the other. $m = 64, n = 1000000$.

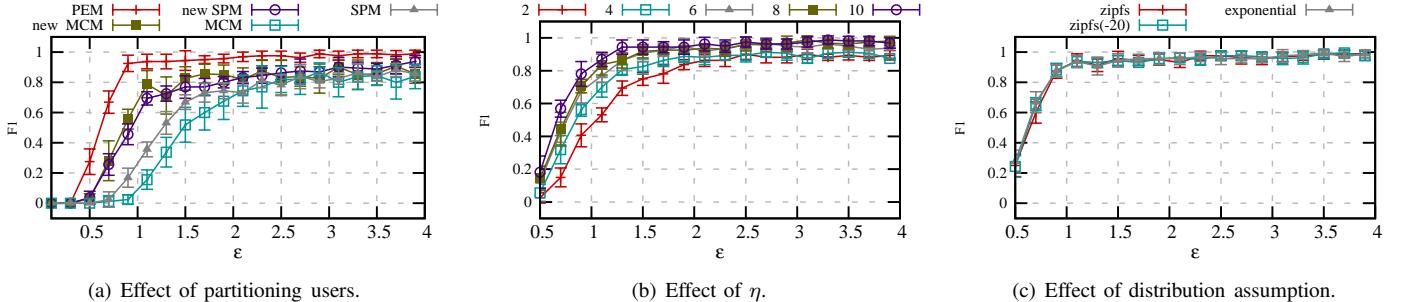


Fig. 8. Evaluation of the synthetic datasets, vary ϵ . $m = 64, n = 1000000$. F1 is plotted.

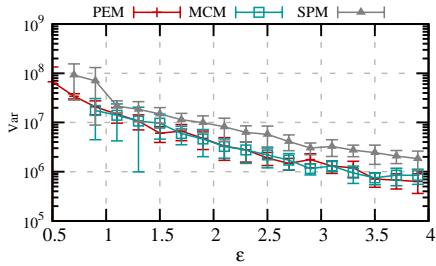


Fig. 9. Evaluation of the synthetic datasets, vary ϵ . $m = 64, n = 1000000$.

which ones are frequent. The problem has been studied in the centralized DP setting [7], [24]. The basic idea is to use sketches to store the heavy hitters and their counts, and then publish the results with some noise. However, in the local setting, it is not even possible to get the candidates. One existing solution is SPM [17]. Hsu et al. [18] and Mishra et al. [25] also provide efficient protocols for heavy hitters, but the error bound is proved higher than MCM, proposed by Bassily and Smith [6]. In this paper, we compare with MCM [6] and SPM [17].

After we finish this work, we also found a simultaneous paper [5] 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. 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. This violates one key observation we made in this paper: the key to improve accuracy is to reduce the number of groups. (Principle 2)

The other method is basically the MCM method with only \sqrt{n} channels (instead of $n^{1.5}$ channels, as suggested in [6]). In our experimental comparison with MCM, we already use around \sqrt{n} channels for MCM, and it significantly underperforms our proposed method. The two methods are proven to provide similar utility guarantees with similar complexities. No experimental comparison with SPM is conducted in [5].

Besides the heavy hitter problem, there are other problems in the LDP setting that rely on mechanisms for frequency estimation. One interesting problem is estimating frequencies of itemsets [15], [16]. Nguyễn et al. [26] studied how to report numerical answers. Chen et al. [9] uses BLH to learn location from users. Wang et al. [29] uses random response and RAPPOR together for learning weighted histogram. Qin et al. [28] estimate frequent items using RAPPOR and BLH, where each user has a set of items. Solutions to these problems can be improved by insights gained in our paper.

Avent et al. [4] propose a system that combines the centralized and local version of DP together and finds heavy hitters. This work is different from ours: The dictionary of heavy hitters is constructed by a group of users who participate in the centralized version of DP. LDP is used only to provide additional information afterwards.

VIII. CONCLUSIONS

In this paper, we propose LDP protocols that finds out heavy hitters in a large domain. The utility of the protocols are thoroughly analyzed and optimized. During analysis, we identify several design principles that can potentially serve as guidelines when solving other LDP problems. Finally, we verify the correctness of analysis and strength of the new methods using empirical experiment on both synthetic and real-world datasets.

Current solutions rely heavily on the distribution. If the distribution is unfavorable, the result will be poor. One interesting problem to explore would be to find protocols that works well for any distribution. Another current limitation is that all the protocol proposed require the domain to be fixed. It would be interesting to find heavy hitters in an unbounded domain. It is also an interesting direction to explore the possibility of using domain-specific knowledge to improve the protocol.

REFERENCES

- [1] “Apple’s ‘differential privacy’ is about collecting your data-but not your data,” <https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/>, 2016.
- [2] “Quantcast top sites,” <https://www.quantcast.com/top-sites/>, 2016.
- [3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.
- [4] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits, “Blender: Enabling local search with a hybrid differential privacy model,” *arXiv preprint arXiv:1705.00831*, 2017.
- [5] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, “Practical locally private heavy hitters,” *arXiv preprint arXiv:1707.04982*, 2017.
- [6] R. Bassily and A. Smith, “Local, private, efficient protocols for succinct histograms,” in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM, 2015, pp. 127–135.
- [7] T.-H. H. Chan, M. Li, E. Shi, and W. Xu, “Differentially private continual monitoring of heavy hitters from distributed streams.” in *Privacy Enhancing Technologies*, vol. 7384. Springer, 2012, pp. 140–159.
- [8] T.-H. H. Chan, E. Shi, and D. Song, “Optimal lower bound for differentially private multi-party aggregation.” in *ESA*. Springer, 2012, pp. 277–288.
- [9] R. Chen, H. Li, A. K. Qin, S. P. Kasiviswanathan, and H. Jin, “Private spatial data aggregation in the local setting,” in *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 2016, pp. 289–300. [Online]. Available: <http://dx.doi.org/10.1109/ICDE.2016.7498248>
- [10] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, “Local privacy and statistical minimax rates,” in *FOCS*, 2013, pp. 429–438.
- [11] C. Dwork, “Differential privacy,” in *ICALP*, 2006, pp. 1–12.
- [12] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *TCC*, 2006, pp. 265–284.
- [13] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014. [Online]. Available: <http://dx.doi.org/10.1561/0400000042>
- [14] Ú. Erlingsson, V. Pihur, and A. Korolova, “Rappor: Randomized aggregatable privacy-preserving ordinal response,” in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 1054–1067.
- [15] A. Evfimievski, J. Gehrke, and R. Srikant, “Limiting privacy breaches in privacy preserving data mining,” in *PODS*, 2003, pp. 211–222.
- [16] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, “Privacy preserving mining of association rules,” in *KDD*, 2002, pp. 217–228.
- [17] G. Fanti, V. Pihur, and Ú. Erlingsson, “Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries,” *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. issue 3, 2016, 2016.
- [18] J. Hsu, S. Khanna, and A. Roth, “Distributed private heavy hitters,” in *International Colloquium on Automata, Languages, and Programming*. Springer, 2012, pp. 461–472.
- [19] K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of ir techniques,” *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.
- [20] N. Johnson, J. P. Near, and D. Song, “Practical differential privacy for sql queries using elastic sensitivity,” *arXiv preprint arXiv:1706.09479*, 2017.
- [21] P. Kairouz, S. Oh, and P. Viswanath, “Extremal mechanisms for local differential privacy,” in *Advances in neural information processing systems*, 2014, pp. 2879–2887.
- [22] N. Li, M. Lyu, D. Su, and W. Yang, *Differential Privacy: From Theory to Practice*, ser. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan Claypool, 2016.
- [23] C. D. Manning, P. Raghavan, H. Schütze et al., *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1, no. 1.
- [24] D. Mir, S. Muthukrishnan, A. Nikolov, and R. N. Wright, “Pan-private algorithms via statistics on sketches,” in *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2011, pp. 37–48.
- [25] N. Mishra and M. Sandler, “Privacy via pseudorandom sketches,” in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2006, pp. 143–152.
- [26] T. T. Nguyễn, 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 preprint arXiv:1606.05053*, 2016.
- [27] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar, “Semi-supervised knowledge transfer for deep learning from private training data,” *arXiv preprint arXiv:1610.05755*, 2016.
- [28] 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.
- [29] S. Wang, L. Huang, P. Wang, H. Deng, H. Xu, and W. Yang, “Private weighted histogram aggregation in crowdsourcing,” in *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 2016, pp. 250–261.
- [30] T. Wang, J. Blocki, N. Li, and S. Jha, “Locally differentially private protocols for frequency estimation.” in *USENIX’17: Proceedings of 26th USENIX Security Symposium on USENIX Security Symposium*. USENIX Association, 2017.

- [31] S. L. Warner, “Randomized response: A survey technique for eliminating evasive answer bias,” *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 63–69, 1965.

APPENDIX

Proof of Proposition 1.

Proof: Suppose we examine the value v_j in round i . We first expand $\text{Piden}_j[i]$ in (5) with $p_j = p \cdot f_j + q \cdot (1 - f_j)$, $\sigma_j^2[i] = n[i] \cdot p_j \cdot (1 - p_j)$, $\mu_j[i] = n[i] \cdot p_j$, $\sigma_0^2[i] = n[i] \cdot q \cdot (1 - q)$, $\mu_0[i] = n[i] \cdot q$, and (according to OLH) $p = \frac{1}{2}$, $q = \frac{1}{e^\epsilon + 1}$:

$$\begin{aligned} & \Phi\left(\frac{\mu_j[i] + \Phi^{-1}\left(\frac{k-j}{N[i]}\right) \cdot \sigma_0[i] - \mu_0[i]}{\sigma_j[i]}\right) \\ &= \Phi\left(\frac{n[i]f_j(p-q) + \Phi^{-1}\left(\frac{k-j}{N[i]}\right) \cdot \sqrt{n[i]q(1-q)}}{\sqrt{n[i](q+f_j(p-q))(1-q-f_j(p-q))}}\right) \\ &= \Phi\left(\frac{\sqrt{n[i]}f_j(p-q)/\sqrt{q(1-q)} + \Phi^{-1}\left(\frac{k-j}{N[i]}\right)}{\sqrt{(q+f_j(p-q))(1-q-f_j(p-q))}/\sqrt{q(1-q)}}\right) \\ &\quad (\text{using } p-q = \frac{e^\epsilon - 1}{2(e^\epsilon + 1)}, q(1-q) = \frac{e^\epsilon}{(1+e^\epsilon)^2}) \\ &= \Phi\left(\frac{\sqrt{n[i]}\frac{f_j(e^\epsilon - 1)}{2e^{\epsilon/2}} + \Phi^{-1}\left(\frac{k-j}{N[i]}\right)}{\sqrt{1 + \left(\frac{e^\epsilon - 1}{e^{\epsilon/2}}\right)^2 \frac{f_j}{2} \left(1 - \frac{f_j}{2}\right)}}\right) \end{aligned} \quad (6)$$

Since $\Phi(\cdot)$ is monotone, when comparing (6) under different settings, it suffices to compare the value inside $\Phi(\cdot)$. Simplify this equation short notations: $A = \Phi^{-1}\left(\frac{k-j}{N[i]}\right) < 0$, $B = \frac{f_j}{2} \left(1 - \frac{f_j}{2}\right)$, $C = \frac{f_j\sqrt{n[i]}}{2}$, and $E(\epsilon) = \left(\frac{e^\epsilon - 1}{e^{\epsilon/2}}\right)^2$. When partitioning users, we have $P_1 = \frac{A}{\sqrt{1+BE(\epsilon)}} + \frac{C}{\sqrt{g/E(\epsilon)+gB}}$; when split privacy budget, we have $P_2 = \frac{A}{\sqrt{1+BE(\epsilon/g)}} + \frac{C}{\sqrt{1/E(\epsilon/g)+B}}$. The goal is to show $P_1 > P_2$.

It is easy to see that $E(\epsilon/g) < E(\epsilon)$, and thus the subtracted term of P_1 is greater than P_2 . As to the first terms, note that in practice, we care more about small f_j , because values with high frequency f_j can always be identified. Therefore, assuming $B \sim 0$, $P_1 > P_2$ if $E(\epsilon/g) < E(\epsilon)/g$ (this is proven by induction on g in Lemma 1). Numerical calculation also validate that $P_1 > P_2$ in most cases. ■

Lemma 1: $E(\epsilon/g) < E(\epsilon)/g$.

Proof: When $\epsilon = 0$, $E(\epsilon/g) = E(\epsilon)/g$. When $\epsilon > 0$, we only needs to show the derivative of the right hand is greater:

define $z = \sqrt{e^\epsilon/g}$.

$$\begin{aligned} & \left(\sqrt{E(\epsilon)/g}\right)' = \sqrt{\frac{1}{g}} \left(z^g - \frac{1}{z^g}\right)' \\ & > \frac{1}{g} (gz^{g-1} + rz^{-g-1}) = z^{g-1} + \frac{1}{z^{g+1}} \\ & > 1 + \frac{1}{z^2} = \left(\sqrt{E(\epsilon/g)}\right)' \end{aligned}$$

The last step is proven by induction. ■

Simpler Joint Estimation for [17]

In SPM, in order to query a value pair from $C_i \times C_j$ where $1 \leq i \neq j \leq g$, maximal likelihood estimation (MLE) is used.

MLE updates every possible combination via a loop, which terminates only if the accumulated update amount is small. This method is slow in practice. Here we introduce a simpler method to recover the joint distribution:

Suppose in the group of n users who report on segments i and j , n_a users have pattern a in segment i , n^b users have pattern b in segment j , and n_a^b users have patterns a and b simultaneously in segments i and j , respectively. The expected “support” this pattern pair a, b receive is

$$E[I_a^b] = n_a^b \cdot p^2 + (n^b - n_a^b) \cdot pq + (n_a - n_a^b) \cdot p q + (n - n^b - n_a + n_a^b) \cdot q^2,$$

where $p = \frac{1}{2}$ and $q = \frac{1}{e^\epsilon + 1}$ are the parameters used in OLH.

Note that we already have partial estimations \tilde{n}_a and \tilde{n}^b on segment i and j , respectively, therefore, we can have the estimated value of n_a^b :

$$\tilde{n}_a^b = \frac{I_a^b - (\tilde{n}_a + \tilde{n}^b) \cdot q(p-q) - n \cdot q^2}{(p-q)^2}$$

Given that \tilde{n}_a and \tilde{n}^b are unbiased, \tilde{n}_a^b is unbiased. In the actual implementation, we use this method.

This method can be extended to multiple (more than two) answers: Let V be a set of answers on different questions (e.g., $V = \{a, b\}$ in the two pattern example),

$$\tilde{n}_V = \frac{I_V - \sum_{i=0}^{n-1} \left[n_{V(i)}^* q^{n-i} (p-q)^i \right]}{(p-q)^n}$$

where $n_{V(i)}^*$ to denote the summation of joint estimation of strings specified by all size i subsets of V , and $\tilde{n}_\emptyset^* = n$.