Differentially Private Summaries for Sparse Data

Graham Cormode, Cecilia Procopiuc, Divesh Srivastava
AT&T Labs–Research
{graham,magda,divesh}@research.att.com

Thanh T. L. Tran
University of Massachusetts, Amherst
ttran@cs.umass.edu

ABSTRACT

Differential privacy is fast becoming the method of choice for releasing data under strong privacy guarantees. A standard mechanism is to add noise to the counts in contingency tables derived from the dataset. However, when the dataset is sparse in its underlying domain, this vastly increases the size of the published data, to the point of making the mechanism infeasible.

We propose a general framework to overcome this problem. Our approach releases a compact summary of the noisy data with the same privacy guarantee and with similar utility. Our main result is an efficient method for computing the summary directly from the input data, without materializing the vast noisy data. We instantiate this general framework for several summarization methods. Our experiments show that this is a highly practical solution: The summaries are up to 1000 times smaller, and can be computed in less than 1% of the time compared to standard methods. Finally, our framework works with various data transformations, such as wavelets or sketches.

General Terms

Algorithms, Security

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]; H.2.0 [**Database Management**]: General—*Security*

Keywords

Differential privacy, anonymization, sparse data

ICDT 2012, March 26–30, 2012, Berlin, Germany.

1. INTRODUCTION

A fundamental problem in data management is how to share datasets without compromising the privacy of the individuals who contribute to the data. A growing science of "anonymization" enables data owners to share sanitized data with the public, external collaborators, or other groups within an organization (the risk of private information leaking guides the amount of masking applied to the data). It also allows businesses to retain detailed information about their customers, while complying with data protection and privacy legislation.

Different techniques are relevant to different threat models: 'syntactic' privacy definitions, such as k-anonymity and l-diversity, preserve more detail at the microdata level, but can be susceptible to attack by determined adversaries [13]. Differential privacy [7] is a more semantic definition, which has gained considerable traction due to its precise privacy guarantees. See recent tutorials on data anonymization for further background and definitions [3, 4, 9].

The original model for differential privacy assumed an interactive scenario: queries are posed to a "gatekeeper," who computes the true answer, adds random noise and returns the result. The random noise is drawn from a particular distribution, whose parameter is determined by the impact any individual can have on the query answer. Other approaches, such as the exponential mechanism [18], also achieve differential privacy and are particularly suited to queries with non-numeric answers.

No matter the privacy mechanism, however, the above scheme can only tolerate a certain number of queries before the "privacy budget" is exhausted and the gatekeeper stops answering. To avoid this restriction, one can release statistics about the data *en masse*. Effectively, the data owner chooses a collection of representative queries to ask, and answers them all with appropriate privacy guarantees. Most often, this is done in the form of histograms or contingency tables, with appropriate noise added to each entry [1]. This is equivalent to releasing the result of groupby/count(*) queries over the original data (with noise).

EXAMPLE 1. *Figure 1(a) shows a table with 5 rows. Its columns represent, respectively, education level (HS = High School Diploma, BS = Bachelor's Degree, PhD = Doctoral Degree), marital status (M = Married, S = Single, D = Divorced), and sex. Figure 1(b) shows the resulting contingency table M and its noisy contingency table M'. Column cnt of M is the actual number of tuples with the respective triplet of attribute values. To meet the requirements of differential privacy, we release a noisy count $M'.ncnt$ for* every triplet *in M. Note that, although cnt is sparse (it has 15 zero-entries), column $M'.ncnt$ is dense. This is because the noise has low probability of being zero.*

tid	Educ	Status	Sex
1	HS	S	M
2	HS	S	M
3	BS	M	F
4	BS	M	F
5	PhD	D	M

(a) Original table with 5 tuples

cid	(Educ,Status,Sex)	cnt	ncnt
1	(HS,S,M)	2	4
2	(HS,S,F)	0	3
3	(HS,M,M)	0	-1
...	...	...	...
10	(BS,M,F)	2	1
11	(BS,D,M)	0	5
...	...	...	...
17	(PhD,D,M)	1	2
18	(PhD,D,F)	0	-3

(b) Contingency table M with column cnt and table M' with column $ncnt$

Figure 1: Table M' is the noisy contingency table for M (column cnt is not published). The density of M is $\rho = 3/18$.

Dataset	Density ρ	Source
OnTheMap	1-5%	`lehd.did.census.gov`
Census Income	0.02-5%	`www.ipums.org`
UCI Adult Data	0.14%	`archive.ics.uci.edu/ml`
Warehouse Data	0.5-2%	proprietary

Table 1: Examples of datasets and their densities.

Scalability Problem. As the above example illustrates, a differentially private (or noisy) contingency table M' can be much larger than the original data. Moreover, M' is also dense, so we cannot represent it compactly (e.g., as a list of non-zero entries).

Let n be the number of non-zero entries in M and m be the total number of rows in M. It is easy to see that if m is computed over attributes $A_1, \ldots, A_k$ of M, then $m = \Pi_{i=1}^{k} |A_i|$, where $|A_i|$ is the number of distinct values of A_i. In other words, m is the *domain size* of M. In Figure 1, $n = 3$ and $m = 3 \times 3 \times 2 = 18$.

Assuming a fixed set of attributes A_i over which M' is computed, we define the *density* of M to be $\rho = n/m$. Hence, applying differential privacy as above generates an output which is about $1/\rho$ times larger than the data size. As we see below, for almost any dataset of reasonable complexity, this makes the method too time and space consuming; in some cases, it is virtually infeasible.

Table 1 shows several examples of widely-used datasets with density in the single-digits or less. (Further details on these datasets, and our density calculations, are in the Appendix.) Thus, a differentially private version of these datasets is 2 to 3 orders of magnitude larger. It is not difficult to see that such examples are the norm, rather than the exception: any data with several attributes A_i of moderately high cardinality $|A_i|$ leads to huge contingency tables of size $m = \Pi_i |A_i| \gg n$. Since modern DBMS's often store tables of hundreds of gigabytes or more, computing a privacy preserving version that is hundreds or thousands of times larger is impractical.

Moreover, the $1/\rho$-factor blow-up also applies to the computation time. Most of the recent results on differential privacy [1, 21, 15] state their running time as (quasi)linear in m, implicitly assuming that $m \approx n$. In fact, as discussed above, this is not the case. For many datasets, $m \gg n$ and the methods are impractical.

In recent years, there have been a few attempts to incorporate differential privacy into data management systems. McSherry's PINQ system [17] adds privacy as an overlay (in the "gatekeeper" model), by automating the addition of noise to query outputs. Closer to the publishing model, Machanavajjhala *et al.* propose a technique tailored for certain types of census data [16]. They design a synthetic data generator based on parameters derived privately from real datasets. The approach produces output data of size comparable to the input size, when direct application of noise mechanisms would generate a dense, noisy output. The synthetic data satisfies a weaker (ϵ, δ)-privacy guarantee, for a larger value of ϵ than what we wish to tolerate. Instead, we aim to produce output directly from the original microdata, with high levels of privacy.

Our Contributions. In this paper we propose a general framework for making differential privacy scalable over sparse datasets, i.e., datasets for which $n \ll m$. Our algorithms compute outputs with strong privacy guarantees, such that the output size is controlled by the data owner. Usually, this is chosen to be close to the size of the original data, or smaller. The running time is proportional to the output size and *independent of* m. In addition, our output has *similar utility* to that of the noisy contingency table. We believe this is a crucial step towards making differential privacy practical in database applications.

Our general approach is as follows. Assume first that we computed the noisy table M'. We can reduce the output size from m to a user-specified size s by returning only a sample of size s from M'. We refer to this sample as the differentially private *summary* of the data. This approach relies on the well-known property that any post-processing of differentially private output remains differentially private [14, 15]. Of course, while this post-process sampling reduces the storage requirements of the output, it does not help the overall computation cost: we must still compute M'. Hence, this approach is effective for the data recipient, but not for the data owner.

Instead, we propose a novel framework: we show how to directly generate the summary from the original data, without computing the huge contingency table M'. This requires some care: The (random) two-step process of generating M', then applying some sampling method over it, implies a probability distribution over possible summaries. Our goal is to provide one-step methods that create summaries with *the same* output distribution, while being much more efficient. We design several one-step summary generation methods, which are provably correct and run in expected time $\Theta(s + n)$, where s is the output size. In general, $s = \Theta(n)$ so our algorithms are linear in the size of the *original data*, but independent of the domain size m.

The algorithms depend on the specific sampling method employed. There is a wealth of work on data reduction techniques. We focus on some of the best known methods, such as filtering and priority sampling [6]. They enable vast quantities of data to be reduced to much more manageable sizes, while still allowing a variety of queries to be answered accurately. We also show how to combine our techniques with data transformations, such as wavelets [21],

sketches [2] and dyadic ranges [11], that can be used to improve query accuracy. To summarize:

- We formalize the problem of computing private summaries of sparse data, and describe appropriate target summaries: filters and samples.
- We show techniques to efficiently generate these summaries, by drawing directly from an implicit distribution over summaries.
- We perform an experimental study over a variety of datasets. We compare the utility of our summaries to that of the noisy contingency tables, by running a large number of queries over both, and comparing their relative accuracies. We conclude that our significantly smaller summaries have similar utility to the large contingency tables, and in some cases, they even improve on it.
- We discuss extensions to combine these techniques with data transformations, such as dyadic ranges, wavelets and sketches.

2. BACKGROUND

Differential Privacy. This privacy model was developed over a series of papers in the 2000s, culminating in Dwork's paper which coined the term [7]. The definition is as follows.

DEFINITION 1 (FROM [7]). *A randomized algorithm K gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on one element (denoted $|D_1 - D_2| = 1$), and all $S \subset Range(K)$:*

$$\mathsf{Pr}[K(D_1) \in S] \leq \exp(\epsilon) \cdot \mathsf{Pr}[K(D_2) \in S]$$

Intuitively, differential privacy guarantees that no individual tuple can significantly affect the released information: the output distribution generated by K is nearly the same, whether or not that tuple is present in the dataset.

For numeric queries, there is a simple recipe for generating differentially private outputs: compute the true answer, then add noise drawn from a specific distribution, as follows.

DEFINITION 2. *Let $q(D)$ be a query over dataset D with numeric output. The* sensitivity of q, *denoted Δ_q, is the maximum change in q when any single tuple of D changes:*

$$\Delta_q = \max_{D_1, D_2 : |D_1 - D_2| = 1} |q(D_1) - q(D_2)|.$$

An ϵ-differentially private mechanism for answering q is to publish $K(D) = q(D) + X$, where X is a random variable as follows:

(i) **Laplace mechanism** *[7]: X is drawn from the Laplace distribution with parameter (ϵ/Δ_q).*

(ii) **Geometric mechanism** *[10]: When $q(D)$ is integer-valued, X can be drawn from the geometric distribution $\mathsf{Pr}[X = x] = \frac{1-\alpha}{1+\alpha}\alpha^{|x|}$, where $\alpha = e^{-\epsilon/\Delta_q}$, $x \in \mathbb{Z}$.*

Both mechanisms achieve ϵ-differential privacy [7, 10]. However, the geometric mechanism adds integer noise, and is usually preferred for queries q with integer answers, e.g., COUNT queries. Therefore we use the geometric mechanism in this paper.

We focus on collections of count queries, such as those required for computing contingency tables. A count query q has sensitivity $\Delta_q = 1$, since adding or removing one individual affects the answer to q by at most one. Moreover, the queries we consider are mutually disjoint (unless otherwise specified); i.e., they are groupby/count(*) queries such that the groupby conditions partition the input dataset. For example, in Figure 1(b), column *cnt* contains the answers to the collection of queries SELECT COUNT(*) FROM *Employees* GROUP BY *Educ*, *Status*, *Sex*. In this case, the sensitivity of the entire collection is 1: adding or removing one individual changes one count value in the collection by one.

Data Reduction. The topic of data summarization is vast, generating many surveys and books on different approaches to the problem [19, 8, 20]. In this paper, we focus on some of the key techniques. We describe them below in the context of applying them to the noisy contingency table M'. Each row i in M' is associated with a value w_i, which depends on its noisy count. Section 3 details how w_i is defined in each case.

Filters are deterministic procedures that prune away parts of the data which are thought to contribute little to the overall query results. The canonical filter is the high-pass filter: data elements with low frequencies (below the filter threshold) are dropped, while the elements with high frequencies (aka the 'heavy hitters') are retained. Applying this to M' means that we retain the rows i for which w_i is above the threshold, and drop the other rows.

Sampling is perhaps the most commonly used data reduction technique. It selects a subset of elements from the input data via a random process. In general, it also assigns a weight to each element in the sample. Queries over the full dataset are then approximated by answering them over the (weighted) sample. The following two sampling procedures have been widely studied.

Threshold sampling [5] is based on a parameter τ. The sample includes each row i with probability $p_i = \min(w_i/\tau, 1)$. If sampled, the sample's weight is assigned to be $\max(w_i, \tau)$. An important property of threshold sampling is that it is unbiased for subset queries (i.e., queries that sum the counts of a subset of rows satisfying a given predicate). That is, the answer is correct in expectation, compared to the answer over the contingency table M'. Since that is also correct in expectation (the noise has mean zero), it follows that the threshold sample of M' returns correct answers in expectation. However, a limitation of threshold sampling is that it does not allow a strict control over the size of the resulting sample: In expectation, the size is $\sum_i p_i = \sum_i \min(w_i/\tau, 1)$, but it may deviate somewhat.

Priority sampling [6] fixes the sample size to be s, and offers strong guarantees about the quality of the sample relative to any other method that generates samples of size exactly s . Each row is assigned a priority $P_i = w_i/r$, where r is drawn uniformly in the range (0,1]. The s elements with the largest P_i values are retained to form the sample. The sample weights are defined based on the $(s+1)$th priority value, denoted τ_s; the adjusted weight of the sampled row i is $\max(w_i, \tau_s)$.

Data Transformations. Methods such as *Fourier or Wavelet transforms* have been widely applied in many areas of computer science. Here, we focus specifically on transforms in the context of privacy. Barak *et al.* [1] manipulate Fourier transforms of contingency tables to ensure integrality and consistency. Xiao *et al.* [21] observe that directly applying the Laplace mechanism (see Definition 2(i)) to contingency tables yields poor answers to range queries, since the error (defined as query variance) grows with the size of the range. Their solution is to add noise in the wavelet domain to provide the same privacy guarantees, while reducing the error for large range queries, due to cancellations. Similar techniques are analyzed by Hay *et al.* [11] using *dyadic ranges* instead of the wavelet transform. A more general approach is given by Li *et al.* [15], who compute query strategies—in the form of linear sums of the input data—that the data owner can release (with appropriate noise) so as to minimize the errors of an expected query workload.

A key difference of these prior results is that they publish output datasets of size $\Omega(m)$. In contrast, we want to publish data of user-

specified size $s \ll m$. In Section 5.2 we discuss how to apply our general framework to these approaches, so they become efficient on sparse data.

Finally, *sketching* techniques summarize the whole data set. The Count sketch hashes each row i to one of B possible buckets via a hash function h, and applies a second hash function g to map each element to either +1 or -1. Each bucket maintains the sum $\sum w_i g(i)$ over all rows i mapped to that bucket by h. Given a target row i, the sum in bucket $h(i)$ multiplied by $g(i)$ is an unbiased estimate for w_i with bounded variance. Taking the mean or median of d independent repetitions further reduces the variance [2]. We describe how to compute private sketches in Section 5.1.

3. COMPUTING PRIVATE SUMMARIES

We now describe how to efficiently compute private summaries for sparse datasets. Recall that a sparse dataset means that its contingency table contains a large portion of zero entries. In other words, if n is the number of non-zero entries in the contingency table M and m is the domain size, or the total number of rows in M, then $n \ll m$. We note that in this setting M can be stored as a list of non-zero entries and design our techniques accordingly. For example, in Figure 1, the compact representation of M consists of rows 1, 10 and 17 of the table in Figure 1(b) (including field *cnt*, but excluding field *ncnt*). Any row not present in this representation is a zero entry. Note that transforming an original table (as in Figure 1(a)) into this compact representation can be done in linear time, and n is at most, but usually smaller than, the size of the original table.

We denote by $M(i)$, resp. $M'(i)$, the count, resp. noisy count, corresponding to row i in table M, resp. M'. With the above convention, i ranges from 1 to m for both tables.

Because the noise can be negative, M' can contain negative entries, which may be undesirable in many applications. A typical adjustment is to replace the negative values by 0. Let M'_+ denote the table obtained from M' via this procedure. For queries with small selectivity (e.g., point queries), using M'_+ tends to be more accurate than M'. However, for large queries it is better to use M', since noise values cancel out. More precisely, the sum of noise values is zero in expectation (but has non-trivial variance).

We present a general framework for computing private summaries for both M' and M'_+. The decision of which contingency table to summarize rests with the data owner, and should depend on the expected query workload over the published summaries.

3.1 The Shortcut Approach

Our aim is to publish M'', a compact summary of M' (or of M'_+). To compute summaries, we need to define a value w_i for each row i; see Section 2. We will either define $w_i = |M'(i)|$ or $w_i = M'(i)$. In the first case, we generate the summmary for M', and in the second case, the summary for M'_+.

The naive (laborious) approach is as follows: Compute the entire M' by first representing M explicitly as a (large, sparse) contingency table, then adding noise to each count to obtain the (large, dense) M'. Afterwards, summarize M' to obtain M''. As discussed in Section 1, this is costly to the point of impracticality, even though M'' is expected to be small, e.g., $|M''| = \Theta(n)$.

Instead, we compute M'' directly from the compact representation of M, without materializing the intermediate table M'. We illustrate this process schematically in Figure 2, which contrasts the *laborious* approach to our proposed *shortcut* approach. The following observation is crucial for designing our algorithms.

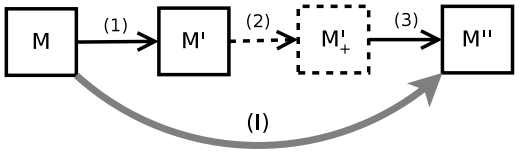


Figure 2: Anonymization process: (1) adding Geometric/Laplace noise, (2) filtering negative entries, (3) sampling, (I) shortcut to generate M'' directly from M.

REMARK 1. *For all summarization techniques we study, each zero entry in M has the same probability p to be chosen in M''. This is because the summarization methods depend only on the values of the counts in M', not their positions; and because the noise added to each zero entry is drawn from the same distribution. Hence, out of the $m - n$ zero entries of M, we include k of them in the summary M'', where k is a random variable that follows the Binomial distribution* $\mathrm{Bin}(m - n, p)$.

In the subsequent sections, we develop shortcut approaches for several summarization methods. Each method requires careful analysis, to show that the resulting M'' is distributed as if it were generated via summarization of M', and that the shortcut approach can be implemented efficiently. We also state how to use the resulting summaries to accurately estimate common queries (subset and range queries), and how to choose parameters such as sample size.

We interchangeably refer to entries in M as items, consistent with sampling terminology. Thus, we say that M has m items, of which n are non-zero. For simplicity, we assume that the values n and $\|M\|_1 = \sum_i |M(i)|$ are not sensitive. Otherwise, it is straightforward to add the necessary noise to mask their true values.

3.2 High-pass Filter

The simplest form of data reduction we consider is to apply a high-pass filter to M': it only retains the large values, and drops the small ones. If we choose an appropriate cut-off to distinguish "small" from "large," the result is to drop a large portion of the data.

We detail the procedure for $w_i = |M'(i)|$, i.e., we compute the summary for M'. The case $w_i = M'(i)$, which computes the summary for M'_+, can be derived in a similar manner.

Let θ be the cut-off value ($\theta \geq 0$). If an item i has value $|M'(i)| \geq \theta$, then we include it in M'', else we drop it. We call this a *two-sided filter* (since it allows both large negative and large positive values). Our approach for generating M'' is to consider the non-zero entries of M separately from the zero-entries. First, in $O(n)$ time, we filter the non-zero entries in M: generate and add noise, then determine whether the resulting value passes the filter. For the $m - n$ zero entries of M, this procedure is too slow, since $n \ll m$. Therefore, we design a statistical process which achieves the same distribution over outputs, without explicitly adding noise to each zero entry, as follows.

Let p_θ denote the probability that a zero entry in M passes the filter θ; p_θ is computed below. We draw the number of entries k that pass the filter from the distribution $\mathrm{Bin}(m - n, p_\theta)$, per Remark 1. We then choose uniformly at random k zero entries in M. For each such entry i, we generate the value $M'(i)$ by drawing noise, *conditional* on the fact that the noise exceeds θ. This may seem like a lot of effort just to generate noise in the output data, but it is a necessary step: we must simulate exactly the output of filtering over the full table M', in order to preserve the differential privacy guarantee. Algorithm FILTER summarizes the algorithm for this shortcut process.

Algorithm FILTER(M)**:** Generates M'' via high-pass filter.

1. For every non-zero entry $M(i)$, add geometric noise with parameter α to get $M'(i)$. If $|M'(i)| \geq \theta$, add $M'(i)$ to M''.
2. For zero entries, sample a value k from the binomial distribution $\mathrm{Bin}(m-n, p_\theta)$, where $p_\theta \triangleq \frac{2\alpha^\theta}{1+\alpha}$.
3. Uniformly at random select k locations i from M such that $M(i) = 0$. For each of these k locations, draw the value of $|M'(i)|$ from the distribution

$$\mathsf{Pr}[|X| \leq x] = (1 - \alpha^{x-\theta+1}).$$

Flip a coin to choose the sign of $M'(i)$, and add $M'(i)$ to M''.

THEOREM 1. *Algorithm* FILTER *generates a summary with the same distribution as the laborious approach under high-pass filtering with parameter $\theta \geq 1$.*

PROOF. Clearly, on the n non-zero entries of M, FILTER acts the same as the laborious approach and thus has the same distribution. We therefore focus on the distribution of entries which are zero in M and are represented in M''. For any entry i such that $M(i) = 0$, the probability that it passes the filter is:

$$\begin{aligned}\mathsf{Pr}[|M'(i)| \geq \theta] &= \sum_{|x|\geq\theta} \frac{1-\alpha}{1+\alpha}\alpha^{|x|} = 2\frac{1-\alpha}{1+\alpha}\alpha^\theta \sum_{x\geq 0}\alpha^x \\ &= 2\frac{1-\alpha}{1+\alpha}\alpha^\theta \frac{1}{1-\alpha} = \frac{2\alpha^\theta}{1+\alpha} \triangleq p_\theta\end{aligned}$$

By Remark 1, the number k of zero entries "upgraded," i.e., which pass the filter after noise is added, follows a binomial distribution $\mathrm{Bin}(m-n, p_\theta)$. Once a zero entry i is chosen, the distribution of its noise $M'(i)$, subject to passing the filter, is:

$$\begin{aligned}\mathsf{Pr}[|M'(i)| = b \mid |M'(i)| \geq \theta] &= \frac{\mathsf{Pr}[|M'(i)| = b]}{\mathsf{Pr}[|M'(i)| \geq \theta]} \\ &= (1-\alpha)\alpha^{b-\theta},\end{aligned}$$

for all $b \geq \theta$; and 0 for all $b < \theta$. The corresponding CDF is:

$$\begin{aligned}\mathsf{Pr}[|M'(i)| \leq x] &= \sum_{b\leq x} \mathsf{Pr}[|M'(i)| = b \mid |M'(i)| \geq \theta] \\ &= \sum_{b=\theta}^{x} (1-\alpha)\alpha^{b-\theta} = 1 - \alpha^{x-\theta+1}, \forall x \geq \theta\end{aligned}$$

Hence the algorithm using this distribution operates as claimed. □

Given this form of the CDF, it is straightforward to draw from it: we draw a uniform random value r in $(0, 1)$, and invert the equation to find for which x is $\mathsf{Pr}[|X| \leq x] = r$. This value of x is then used as the value $|M'(i)|$; after flipping a coin for the sign, we add either x or $-x$ to the summary M''.

The *one-sided* filter, which passes values $M'(i) \geq \theta$, is very similar. The only differences are that p_θ is now equal to $\frac{\alpha^\theta}{1+\alpha}$, and the CDF is $\mathsf{Pr}[X \leq x] = 1 - \alpha^{x-\theta+1}$. So we draw from the same distribution as before, but no longer flip a coin for the sign.

Time and Space Analysis. The running time of this algorithm consists of the time to process the n non-zero entries of M, which is $O(n)$; and the time to upgrade the k zero entries.

Rejection sampling. Selecting the locations of the k zero entries requires some care, since M is stored as a list of non-zero entries. We perform the following *rejection sampling:* uniformly at random pick a location $i \in \{1, \ldots, m\}$, and accept it if $M(i) = 0$; otherwise, reject and repeat. The number of trials per selected location follows the geometric distribution with success probability $(m-n)/m$. Thus, the expected number of trials to choose one zero entry is $m/(m-n) \leq 2$, since $n \ll m$. Checking whether $M(i) = 0$ for a given i also takes (expected) constant time, if M is stored as a hash table of non-zero entries. Thus, selecting one zero entry takes expected time $O(1)$. Drawing a noise value for it also takes constant time. The expected time to upgrade all zero entries is $O(k)$, and the overall expected time is $O(n+k)$.

The value k depends on p_θ, which in turn depends on θ. More precisely, $\mathsf{E}[k] = (m-n)p_\theta$. Since $m-n$ is assumed to be large, k is tightly concentrated around its expectation. Suppose we have a target output size of s tuples; e.g., $s = \Theta(n)$. How should we choose θ? Assume we can tolerate an output of size at most $s + n = \Theta(n)$. It is thus sufficient to focus on the zeros: if we pick $p_\theta \approx s/(m-n)$, then we expect s zeros to be upgraded. This leads us to pick

$$\theta = \frac{\log(\frac{(1+\alpha)s}{2(m-n)})}{\log \alpha},$$

for two-sided filters. We apply this threshold, and consider the output of the FILTER algorithm: if it is sufficiently close to s, we accept the output. Otherwise, we choose a higher θ. Rather than re-running the algorithm, we can simply take its output and apply a higher filter θ' on it until we reach the desired sample size. If the output of the algorithm is too small, we re-run the FILTER algorithm with the same θ value until we obtain a sufficiently large summary. Since $\mathrm{Bin}(m-n, p_\theta)$ is highly concentrated around its mean, it is highly likely that we generate a summary of (close to) the desired size s in only a constant number of such trials.

Query Answering. Naively, the filtered output can be used directly to answer queries. This is clearly biased, since small values in M' are replaced with zero. However, it may work well in practice if θ is chosen so that values below θ are mostly noise, while most of the original non-zero values in M were comfortably above θ. One can also consider other heuristic corrections to the output, such as assuming that all values absent from M'' are, e.g., $\theta/2$ or some other calibrated value. The subsequent methods we consider avoid this issue by providing unbiased estimators for query answering.

3.3 Threshold and Priority Sampling

More sophisticated data reduction techniques are based on sampling. In this section and the next, we discuss how to generate M'' as a random sample of M', without explicitly generating M'. There are many sampling procedures. The simplest is to uniformly sample items in M, then create M'' by adding noise to them. This is easy to implement, but unlikely to have much utility: since M is sparse, we would sample almost exclusively entries with $M(i) = 0$, making the sample virtually useless. Instead, we extend the intuition from the high-pass filter: items in M' with high (noisy) values are more likely to correspond to non-zero entries in M, i.e., to original data. We should include them in our sample with higher probability. The filtering approach achieved this deterministically: items below the threshold had zero chance of being included. When sampling, we now allow every item a chance of being in the summary, but set the probability proportional to its (noisy) count.

In the following, we provide a detailed description of how to sample from M' (i.e., we set $w_i = |M'(i)|$). A similar procedure can be used for sampling from M'_+; the formulas are in fact easier to derive in that case, so we omit them here.

Threshold Sampling. Let $w_i = |M'(i)|$. The threshold sampling procedure [5] with parameter τ is as follows: include item i in the

sample with probability $p_i = \min(\frac{|M'(i)|}{\tau}, 1)$. This means that truly heavy items that have $|M'(i)| > \tau$ are included in the sample with probability 1.

Algorithm THRESHOLD**.** Generate M'' via threshold sampling.

1. For every non-zero entry in $M(i)$, add geometric noise to get $M'(i)$ and add it to M'' with probability $p_i = \min(\frac{|M'(i)|}{\tau}, 1)$.
2. For the zero entries, sample a number k from the binomial distribution $\mathrm{Bin}((m-n), p_\tau)$, where
$$p_\tau \triangleq \frac{2\alpha(1-\alpha^\tau)}{\tau(1-\alpha^2)}$$
3. Uniformly at random select k entries i from M such that $M(i) = 0$. For each of these k entries, draw the value of $M'(i)$ from the distribution $\mathsf{Pr}[X \le \nu]$ given by:
$$\begin{array}{ll} \tau\alpha^{-\nu}C_\tau(1-\alpha), & \text{if } \nu \le -\tau \\ C_\tau(-\nu\alpha^{-\nu} + (\nu+1)\alpha^{-\nu+1} - \alpha^{\tau+1}), & \text{if } -\tau < \nu \le 0 \\ \frac{1}{2} + \alpha C_\tau(1-(\nu+1)\alpha^\nu + \nu\alpha^{\nu+1}), & \text{if } 0 < \nu \le \tau \\ \frac{1}{2} + \alpha C_\tau(1-\alpha^\tau - \tau\alpha^\nu(1-\alpha)), & \text{if } \nu > \tau \end{array}$$
where $C_\tau = \frac{1}{2\alpha(1-\alpha^\tau)}$ is a constant depending on τ, α. Add $M'(i)$ to M''.

THEOREM 2. *Algorithm* THRESHOLD *generates a summary with the same distribution as the laborious approach under threshold sampling with parameter $\tau > 0$.*

PROOF. The non-zero entries are sampled with the probabilities defined for threshold sampling, so both approaches have the same distribution on the n non-zero entries of M.

Let S be the set of zero entries included in M''. A zero entry i is included with probability $p_i = \min(\frac{|M'(i)|}{\tau}, 1)$. Then:
$$\mathsf{Pr}[i \in S | M'(i) = \nu] = \min(\tfrac{|\nu|}{\tau}, 1)$$
The next two summations follow by standard algebra:
$$\sum_{x=0}^{v} \alpha^x = \frac{1-\alpha^{v+1}}{1-\alpha} \quad (1)$$
$$\sum_{x=0}^{v} x\alpha^x = \frac{\alpha}{(1-\alpha)^2}(1-(v+1)\alpha^v + v\alpha^{v+1}) \quad (2)$$
Using (1) and (2), the probability that zero entry i is included is:
$$\begin{aligned} \mathsf{Pr}[i \in S] &= \sum_\nu \mathsf{Pr}[i \in S | M'(i) = \nu]\mathsf{Pr}[M'(i) = \nu] \\ &= \sum_{|\nu| \le \tau} \frac{|\nu|}{\tau}\frac{1-\alpha}{1+\alpha}\alpha^{|\nu|} + \sum_{|\nu|>\tau} \frac{1-\alpha}{1+\alpha}\alpha^{|\nu|} \\ &= 2\left(\frac{1-\alpha}{\tau(1+\alpha)}\sum_{\nu=0}^{\tau}\nu\alpha^\nu + \sum_{\nu>\tau}\frac{1-\alpha}{1+\alpha}\alpha^\nu\right) \\ &= 2\left(\frac{\alpha}{\tau(1-\alpha^2)}(1-(\tau+1)\alpha^\tau + \tau\alpha^{\tau+1}) + \frac{\alpha^{\tau+1}}{1+\alpha}\right) \\ &= \frac{2\alpha(1-\alpha^\tau)}{\tau(1-\alpha^2)} \triangleq p_\tau \end{aligned}$$

By Remark 1, the number of upgraded zeros follows the binomial distribution $\mathrm{Bin}(m-n, p_\tau)$. Given that i is chosen, its value $M'(i)$ is conditioned on the fact that it was sampled:
$$\begin{aligned} \mathsf{Pr}[M'(i) = \nu | i \in S] &= \frac{\mathsf{Pr}[i \in S | M'(i) = \nu]\mathsf{Pr}[M'(i) = \nu]}{\mathsf{Pr}[i \in S]} \\ &= \frac{\min(\frac{|\nu|}{\tau}, 1) \cdot \frac{1-\alpha}{1+\alpha}\alpha^{|\nu|}}{p_\tau} \end{aligned}$$
Substituting p_τ in the above, we have:
If $|\nu| > \tau$, $\mathsf{Pr}[M'(i) = \nu | i \in S] = \frac{\tau(1-\alpha)^2\alpha^{|\nu|}}{2\alpha(1-\alpha^\tau)}$
If $|\nu| \le \tau$, $\mathsf{Pr}[M'(i) = \nu | i \in S] = \frac{(1-\alpha)^2|\nu|\alpha^{|\nu|}}{2\alpha(1-\alpha^\tau)}$
Let $C_\tau = \frac{1}{2\alpha(1-\alpha^\tau)}$; then the CDF of this distribution, at any ν, can be computed by summing the probability of $\mathsf{Pr}[M'(i) = x | i \in S]$, $\forall x \le \nu$, given by the above two expressions. It can be broken into four pieces and written as follows.
$$\begin{aligned} \mathsf{Pr}[X = \nu \le -\tau] &= \tau\alpha^{-\nu}C_\tau(1-\alpha) \\ \mathsf{Pr}[-\tau < X = \nu \le 0] &= C_\tau(\tau\alpha^\tau(1-\alpha) - \nu\alpha^{-\nu} - \tau\alpha^\tau \\ &\quad + (\nu+1)\alpha^{-\nu+1} + (\tau-1)\alpha^{\tau+1}) \\ &= C_\tau(-\nu\alpha^{-\nu} + (\nu+1)\alpha^{-\nu+1} - \alpha^{\tau+1}) \\ \mathsf{Pr}[0 < X = \nu \le \tau] &= \tfrac{1}{2} + \alpha C_\tau(1-(\nu+1)\alpha^\nu + \nu\alpha^{\nu+1}) \\ \mathsf{Pr}[\tau < X = \nu] &= \tfrac{1}{2} + C_\tau(\alpha(1-(\tau+1)\alpha^\tau + \tau\alpha^{\tau+1}) \\ &\quad + \tau\alpha^{\tau+1}(1-\alpha^{\nu-\tau})(1-\alpha)) \\ &= \tfrac{1}{2} + \alpha C_\tau(1-\alpha^\tau - \tau\alpha^\nu(1-\alpha)) \end{aligned}$$
So picking $M'(i)$ via this CDF has the correct distribution. □

Time and Space Analysis. It follows immediately from the description of the algorithm that the sample can be generated in expected time $O(n+k) = O(n + p_\tau(m-n))$. (For step 3, we use rejection sampling as in Section 3.2). We now analyze how to choose τ given a desired sample size s. We could proceed as in Section 3.2. However, we can make a more accurate estimation by taking into account the statistics of the non-zeros, as follows. The expected threshold sample size is $t = \sum_i \min(|M'(i)|/\tau, 1)$. For large τ, this can be approximated by
$$\sum_i \frac{|M'(i)|}{\tau} \le \frac{1}{\tau}\sum_i (|M(i)| + |G(\alpha)|)$$
where $G(\alpha)$ is the geometric distribution with parameter α. Then
$$\mathsf{E}[|G(\alpha)|] = 2\sum_{j=0}^{\infty}\frac{1-\alpha}{1+\alpha}j\alpha^j = \frac{1-\alpha}{1+\alpha}\frac{2\alpha}{(1-\alpha)^2} = \frac{2\alpha}{1-\alpha^2},$$
using (2) (for $v \to \infty$). Hence, in expectation this approximation is $\|M\|_1/\tau + m\mathsf{E}[|G(\alpha)|]/\tau = (\|M\|_1 + 2m\alpha/(1-\alpha^2))/\tau$.

Rearranging this suggests that we should pick τ in the region of $\frac{1}{s}(\|M\|_1 + 2m\alpha/(1-\alpha^2))$ for a desired sample size s. As with the high-pass filter, we can draw a sample based on an estimate of τ, then refine our choice of τ to achieve a desired sample size. However, this is taken care of more directly when we build a priority sample of fixed size, as discussed below.

Query Answering. Each element is sampled with probability $p_i = \min(|M'(i)|/\tau, 1)$. For each sampled i, we adjust its value in M'' to $M'(i)/p_i$. This yields an unbiased Horvitz-Thompson estimator for subset queries [5]. The data owner can scale all values before release. This allows queries to be answered by evaluating them over the sample, using the adjusted values. However, for very small queries (e.g., point queries), it can be more accurate to use the (biased) estimator of the unadjusted values.

Priority Sampling has been advocated as a method to generate a sample of fixed size, with strong accuracy properties [6]. The sampling scheme is defined as follows. Each entry is assigned a priority $P_i = \frac{|M'(i)|}{r_i}$, where r_i is a random value chosen uniformly from the range (0,1]. The scheme draws a sample of size s by

picking the items with the s largest priorities, and also retaining the $(s+1)$th largest priority for estimation purposes.

To efficiently build a priority sample of M', suppose that we knew τ_s, the $(s+1)$th largest priority. Then $M'(i)$ is sampled if

$$P_i = \frac{|M'(i)|}{r_i} > \tau_s \iff r_i < \frac{|M'(i)|}{\tau_s}.$$

Since r_i is uniform over (0,1], the probability of this event is $\min(|M'(i)|/\tau_s, 1)$. In other words, this procedure can be seen as equivalent to threshold sampling with threshold τ_s. (Therefore, we omit its full description here.) We next discuss how to choose τ_s. The crucial difference from threshold sampling is that τ_s is data-dependent, so we do not know it in advance. However, we can *guess* a good value: one which will yield a sample of size $s' = \Theta(s)$, where the constant hidden by the Θ notation is small (but ≥ 1). We first use our guess for τ_s to draw a corresponding threshold sample. We then augment each item in the resulting sample with an assigned priority r_i. That is, conditional on item i being in the sample, we draw an r_i consistent with this outcome. For i, we must have $0 < r_i \leq |M'(i)|/\tau_s$ (else i would not have passed the threshold), but beyond this there are no additional constraints, so r_i is picked uniformly in the range $(0, |M'(i)|/\tau_s]$, and subsequently its value is held fixed.

Given these priorities for a sample of size $s' = \Theta(s)$ (which is distributed exactly as the s' largest priorities over all of M'), we reduce the sample size to s by picking the s largest priorities (and retaining the $(s+1)$th priority). The running time is $O(s') = O(s)$ (in expectation). The result has the desired distribution; it follows from the above discussion and results on threshold sampling.

THEOREM 3. *A priority sample of M' of size s can be generated via the shortcut approach in expected time $O(s+n)$.*

Query Answering. As with threshold sampling, we adjust the sampled values for query answering: letting τ_s be the $(s+1)$th priority, we assign each item a value with magnitude $\max(\tau_s, |M'(i)|)$ while keeping the sign of $M'(i)$. This is unbiased, and has lowest variance for arbitrary subset queries compared to any other scheme that draws a sample of s items [6].

3.4 Combining Sampling with Filtering

We have argued that both sampling and filtering are useful ways to summarize data. In particular, filtering removes small counts which are most likely noise. Setting the threshold θ too low removes a lot of noise, but will still pass too many items, while setting it too high will remove too many true data items. A natural compromise is to combine sampling with filtering: generate M'' from M' by first filtering out low values, then sampling over the result. We expect this to give us the best properties of both methods: noise removal and bounded output size. For clarity, we describe the combination of high-pass filter and threshold sampling. We can support filter and priority sampling via the observation in Section 3.3, that we can extract a priority sample from a threshold sample. In practice, both combinations perform similarly. In Section 4 we report results only on filtering and priority sampling, which achieves output size exactly s.

There are two parameters: θ for filtering and τ for sampling. When $\tau \leq \theta$, every item which passes the filter enters the sample, i.e., we have just filtering. When $\theta = 0$, every item passes the (two-sided) filter, i.e., we have just sampling. The interesting range occurs when $0 < \theta < \tau$: items in M' with absolute weights below θ are dropped, above τ are always included in M'', and in between are included with probability proportional to their weight.

As in previous cases, the non-zero elements of M are handled directly: we add noise drawn from the geometric distribution to obtain $M'(i)$, and first filter then sample the element. It is the large number of zero elements which we must treat more carefully to keep the process efficient.

We follow the outline from previous sections. Let S be the set of zero entries selected in M'', and let $p_{\theta,\tau} = \mathsf{Pr}[i \in S]$ be the probability that the zero entry i is selected, i.e., it passes both the filter and the sampling. Then

$$\begin{aligned}
p_{\theta,\tau} &= \sum_{|\nu|\geq\theta} \mathsf{Pr}[i \in S | M'(i) = \nu]\mathsf{Pr}[M'(i) = \nu] \\
&= \sum_{|\nu|>\tau} \frac{1-\alpha}{1+\alpha}\alpha^{|\nu|} + \sum_{\theta\leq|\nu|\leq\tau} \frac{1-\alpha}{1+\alpha}\alpha^{|\nu|}\frac{|\nu|}{\tau} \\
&= 2\left(\sum_{\nu>\tau} \frac{1-\alpha}{1+\alpha}\alpha^{\nu} + \sum_{\theta\leq\nu\leq\tau} \frac{1-\alpha}{1+\alpha}\alpha^{\nu}\frac{\nu}{\tau}\right) \\
&= 2\left(\frac{\alpha^{\tau+1}}{1+\alpha} + \frac{\alpha}{\tau(1-\alpha^2)}(\theta\alpha^{\theta-1} - (\theta-1)\alpha^{\theta}\right. \\
&\qquad \left. -(\tau+1)\alpha^{\tau} + \tau\alpha^{\tau+1})\right) \\
&= \frac{2}{\tau(1-\alpha^2)}(\theta\alpha^{\theta} - (\theta-1)\alpha^{\theta+1} - \alpha^{\tau+1})
\end{aligned}$$

Observe that for the case $\theta = 0$, this simplifies to $\frac{2\alpha(1-\alpha^{\tau})}{\tau(1-\alpha^2)}$, i.e., the expression for p_τ for threshold sampling in Section 3.3. We draw from $\mathrm{Bin}(m-n, p_{\theta,\tau})$ to determine the number of zero entries from M to sample. Given such a location i, conditioned on it being included in the sample, the distribution of its value is now

$$\begin{aligned}
\mathsf{Pr}[M'(i) = \nu | i \in S] &= \frac{\mathsf{Pr}[i \in S | M'(i) = \nu]\mathsf{Pr}[M'(i) = \nu]}{\mathsf{Pr}[i \in S]} \\
&= \tau C_{\theta,\tau}(1-\alpha)^2\alpha^{|v|}\min(\frac{|\nu|}{\tau}, 1)
\end{aligned}$$

where the constant $C_{\theta,\tau} = (2(\theta\alpha^{\theta} - (\theta-1)\alpha^{\theta+1} - \alpha^{\tau+1}))^{-1}$. The CDF of the sampled values has four cases, $\mathsf{Pr}[X \leq \nu] =$

$$\begin{cases}
\tau C_{\tau,\theta}(1-\alpha)\alpha^{-\nu}, & \text{if } \nu \leq -\tau \\
C_{\tau,\theta}(-\nu\alpha^{-\nu} + (\nu+1)\alpha^{-\nu+1} - \alpha^{\tau+1}), & \text{if } -\tau < \nu \leq -\theta \\
\frac{1}{2} + C_{\tau,\theta}(\theta\alpha^{\theta} - (\theta-1)\alpha^{\theta+1} & \\
\qquad -(\nu+1)\alpha^{\nu+1} + \nu\alpha^{\nu+2}), & \text{if } \theta \leq \nu \leq \tau \\
1 - \tau C_{\tau,\theta}(1-\alpha)\alpha^{v+1}, & \text{if } \nu > \tau
\end{cases}$$

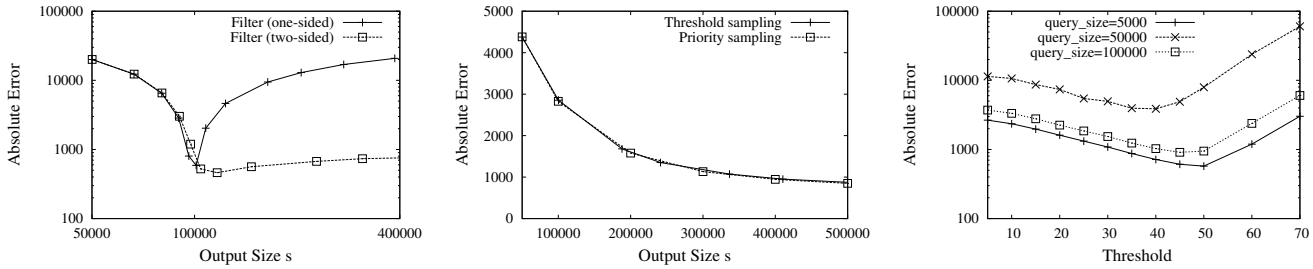
Query Answering. We adjust the values in M'' the same way as for threshold sampling. This provides unbiased estimates for subset queries over the underlying data. In this case, the underlying data is the filtered version of M'. Since filtering with a low threshold removes much of the noise, being unbiased over the filtered M' may be a good approximation of the original M data.

4. EXPERIMENTAL STUDY

4.1 Experimental Setup

We evaluate our methods on a mixture of real and synthetic data. We first use synthetic data, where we can observe the impact of varying data characteristics on the quality of summaries produced. We then report experiments on two real datasets in Section 4.4.

Default Setup. Unless otherwise specified, we use a synthetic dataset generated as follows. We set the domain size to $m = 10^6$, so that it is feasible to apply the standard geometric mechanism,



(a) Filtering: Varying sample size (b) Threshold and Priority sampling (c) Filter-Priority sampling

Figure 3: Impact of parameter choice for each method

which materializes the noisy contingency table M' of size m. This is included as a baseline to which we compare our techniques, though it is not practical for the sparse datasets of interest. Table M is generated as a list of non-zero entries. The non-zero data entries are drawn from a Gaussian distribution with mean $\mu = 100$, standard deviation $\sigma = 20$, and they are then rounded to the nearest positive integer. The default data density is $\rho = 0.1$ (hence, $n = 10^5$), and the non-zero values are uniformly distributed in the domain.

The default privacy parameter is $\epsilon = 0.1$. Experiments were run on a machine with Intel Xeon 3GHz processor and 4GB of memory. Each accuracy result was averaged from 50 runs where the data is generated with different random seeds.

We measure utility by computing the absolute errors for various queries, i.e., the L_1 difference between the true query answer (on M) and that estimated from the published data (M''). To compare the utility of the resulting summaries, we often discuss the relative error of a set of queries, computed as the sum of absolute errors divided by the sum of true answers. This indicates the general performance, while avoiding distortion of results by queries whose true answer is zero or near-zero. We focus on the important class of subset queries, which estimate the sum of counts associated with a subset of locations in M. For each query of size r, we choose r random locations in $\{1, \ldots, m\}$. We report results averaged over $O(m)$ queries.

4.2 Filtering and Sampling Parameter Setting

Figure 3 shows the observed error as we vary the parameters θ for filtering and τ for threshold sampling, which affects the sample size. For clarity, we plot absolute errors, since the relative errors are small (see below).

Filtering. We compare the two high-pass filters: one-sided (pass only positive values above θ) and two-sided (pass all items whose absolute value is above θ). We fix the subset size of queries to 5000 and adjust the filtering threshold θ to get different summary sizes. Figure 3(a) shows the accuracy on a log scale, as it varies by several orders of magnitude. Two-sided filtering is generally more accurate than one-sided filtering. When most of the (originally) non-zero data points are in the sample, query answers using two-sided filtering estimates are unbiased. In this case the relative error is very low: consistently around 1%. One-sided filtering is more likely to over-estimate because some data points which were originally zero are now positive entries in the sample. When the summary size is small, we expect it to be dominated by the non-zeros and hence both techniques have similar accuracy. The minimal error occurs when the sample contains most of the true data points and few originally zero entries. But this may require a θ value that results in a larger than desired summary. As we observed the same behavior across a variety of query sizes and datasets, we henceforth only use two-sided filtering, which is more robust.

Threshold and priority sampling. Figure 3(b) shows the accuracy of threshold sampling and priority sampling, over queries with subset size 5000. The parameter τ for threshold sampling is set to get a desired sample size s (see Section 3.3). This corresponds to relative errors in the range 1% to 7%. The behavior of threshold sampling is very similar to that of priority sampling, as expected from our analysis. The only difference is that priority sampling returns a sample with fixed size s as required, but needs more time to find the $(s+1)$th priority and adjust the weights accordingly, as shown in Figure 4. Since threshold and priority sampling behaved similarly across all other query sizes we tested, we show only the latter in the below experiments.

Filter-priority sampling. Figure 3(c) shows the accuracy for the combination of two-sided filtering with priority sampling (which we dub *Filter-Priority*) for θ between 5 and 70. We fix the priority sample size to $s = 10^5$, equal to the cardinality of the original input data. The relative errors measured are within [0.3%, 6%]. For this setting, we find that $\theta = 40$ is best, which is in the region of half of the mean μ of the non-zero data values. In general, setting the threshold θ should filter most of the (upgraded) zeros while retaining a large number of the original non-zero data. For queries on larger subsets, we set θ smaller to ensure more non-zeros appear in the sample, so we rely more on the unbiased properties of priority sampling. The choice of θ is also affected by the data density.

4.3 Comparing Summarization Techniques

We next compare the performance of our techniques to the baseline method, denoted *Geometric*, of generating M' by adding geometric noise to all its m entries.

Running time. Our main motivation for this research was that methods like the geometric mechanism over contingency tables do not scale well for sparse data. To illustrate this scalability issue, we measure the time to create a summary of input data of cardinality $n = 10^6$ as we vary the domain size m, obtaining different data density values, from 10^{-1} to 10^{-5}. This spans the range of densities mentioned in Section 1. The sample size is $s = n$.

Figure 4 shows the running time of *Geometric*, as well as that of our various shortcut approaches. We also include the *Wavelet* approach, discussed in more detail in Section 5. Among all methods, *Filter* is the fastest, while *Priority* and *Filter-Priority* have a higher cost due to the extra effort to find the $(s+1)$th priority and adjust the weights. Under the very densest setting, $\rho = 0.1$, the running time is about the same for all methods. However, as the data becomes more sparse, all our techniques are orders of magnitude faster than *Geometric*. This demonstrates that the standard

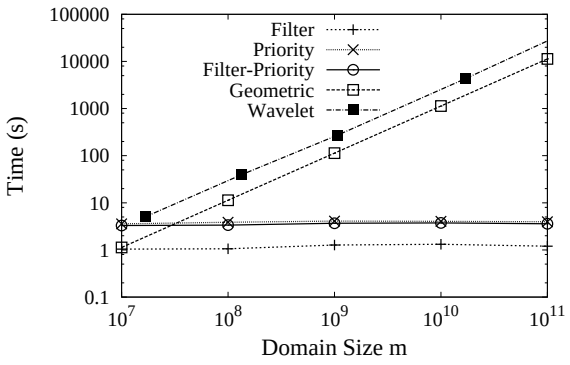


Figure 4: Running time when varying the domain size

approach, which materializes the huge table M', is infeasible in most cases. The wavelet approach also processes every cell in the domain of M, and fails to scale for large domains.

Query accuracy. We create summaries of size $s = n = 10^5$ over the default dataset. Figure 5(a) shows the accuracy of *Filter, Priority, Filter-Priority,* and *Geometric,* for both medium and large subset queries over the default dataset. Apart from *Priority,* all techniques have relative error between 5-10% for subsets of size 100, and this decreases as the subset size increases.

Note that *Geometric* publishes the entire noisy table M', while our techniques release much smaller summaries of M'. Thus, we use the accuracy of *Geometric* as a benchmark against which we evaluate our summaries. For small and medium size queries, *Filter* with $\theta = 50$ performs best, while *Priority* is more accurate when the query touches more than 10,000 entries. *Filter-Priority* (here using $\theta = 40$) combines the advantages of the two techniques, having accuracy slightly better than *Geometric*.

This is a somewhat surprising outcome: at best, we hoped to equal the accuracy of the (impractical) geometric mechanism with a compact summary; but here and in the next section, we see examples with *better* accuracy. The reason is that summarization helps us: the noise introduced by the privacy mechanism is less likely to reach the output summary, so we get somewhat more accurate answers. This does not alter privacy guarantees—as the user is free to do any post-processing of the output of the geometric mechanism, such as apply a filter to it—rather, this indicates that the summarization techniques maintain the same privacy and can improve the utility of the released data.

Figure 5(b) shows the corresponding plot over a different dataset: the non-zero entries are drawn from a Gaussian with higher standard deviation $\sigma = 40$. *Filter* performs less well, as it is harder to separate the true data from the noise introduced in M'. In this experiment, the summary techniques are generally less accurate than *Geometric*. The gap is narrowed for larger summaries: picking a sample size of 2×10^5 (twice as large) was sufficient to make the accuracy of *Filter-Priority* comparable to that of *Geometric*.

Impact of data density. Next, we reduce the data density to $\rho = 0.01$ while using the same domain size $m = 10^6$; hence, $n = 10^4$. We also set the sample size $s = n$. Figure 5(c) shows that *Filter* outperforms *Priority* and *Filter-Priority* for most sample sizes: since there are more zero entries in M, the probability of a zero appearing in the sample increases, and the proportion of sampled non-zero values decreases. But *Filter-Priority* (with $\theta = 50$) still has higher accuracy than *Geometric* across the whole spectrum of subset sizes, due to efficiently canceling most of the noise from zero entries. This result suggests that, across a range of datasets and query sizes, this combined *Filter-Priority* method has the best of all worlds: it uses filtering to prune away pure noise, and sampling to favor more significant data entries without bias, and it produces a summary of fixed size. This also confirms that our techniques can create accurate summaries for data of low density.

4.4 Evaluation on Real Data

Having studied the performance of our techniques on synthetic data where we can vary parameters, we now apply them to two real datasets.

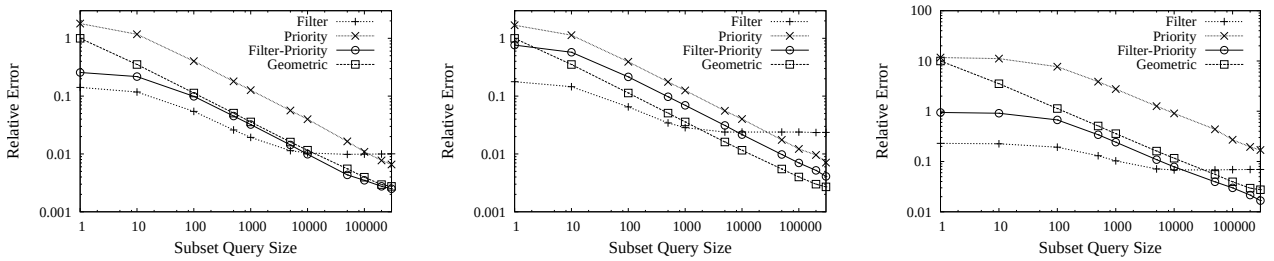
OnTheMap Data. The "OnTheMap" data, described in the Appendix, is actually synthesized from a real dataset on commuting patterns. It is designed to represent realistic commuting patterns. For the purposes of our study, we treat it as the sensitive table M and compare our techniques for protecting the privacy of these (synthetic) individuals. For this data, the domain size m is about 2.2×10^9, and the data size n is 1.7×10^7. We aim to publish a sample with size $s = n$ for our methods, and compare them with the *Geometric* approach. Figure 6(a) shows that our summary methods are up to an order of magnitude faster. Equally significant, Figure 6(b) shows that the output size of our summaries is bounded, meeting the target $s = n$; while the output size of *Geometric*, equal to m, is hundreds of times larger.

Figure 6(c) shows the corresponding accuracy for summaries of size 3×10^6 on an aggregated version of the data. We observe that *Filter* is accurate for small subset queries, but becomes inaccurate for larger ones, This is due to the high variance in the magnitude of non-zero entries: deleting too many small values actually wipes out a lot of the original data. *Priority* and *Filter-Priority* with a small threshold have better accuracy, comparable to *Geometric* for most queries, while generating more compact summaries. In terms of relative errors, all queries for subsets of 5% or more of the data have errors less than 0.8%.

We also compared to the method of Machanavajjhala *et al.* [16] to anonymize data via synthetic data generation. We applied their method using parameters $\epsilon = 1$, $\delta = 10^{-4}$, to satisfy (ϵ, δ)-probabilistic differential privacy. While this approach has been shown to approximately preserve features such as the distribution of commuting distance in the data [16], it does not seem to help in accurately answering our queries. For example, over the same set of queries we observe absolute error over three times larger than for *Geometric* with the same privacy parameters. Furthermore, it does not seem possible to find a setting of the parameters $\alpha(i)$ required by the method to obtain stricter privacy guarantees (i.e., $\epsilon < 1$). By comparison, our experiments are carried out with the even stronger privacy guarantee of $\epsilon = 0.1$.

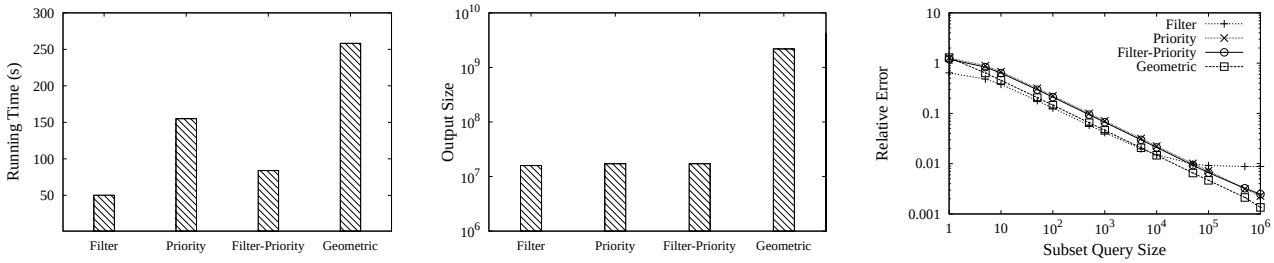
Census Income Data. We next consider the census income dataset described in the Appendix. Figures 7(a) and 7(b) show the running time and the output size under a fine-grained gridding where income is rounded to every thousand. The resulting data set has domain size $m = 1.6 \times 10^{10}$ and cardinality $n = 3.8 \times 10^6$. For this sparse data, our sampling and filtering techniques are faster than *Geometric* by about two orders of magnitude, and publish a significantly smaller output.

Figure 7(c) shows the accuracy of the different techniques for a slightly coarser gridding. For this data, queries have to touch quite a large number of entries before they become accurate. Methods which involve a filter step perform well on this data, and are substantially more accurate than the baseline *Geometric* method for nearly all query sizes.



(a) Subset queries, (μ=100, σ=20, ρ=0.1) (b) Subset queries, (μ=100, σ=40, ρ=0.1) (c) Subset queries, (μ=100, σ=20, ρ=0.01)

Figure 5: Experimental results on subset queries



(a) Running time (b) Output size (c) Accuracy

Figure 6: Experimental results using "OnTheMap" data

5. DATA TRANSFORMATIONS

In this section, we discuss how our methods are compatible with various popular data transformations: randomized sketching and dyadic/wavelet transforms for range queries.

5.1 Sketch Summarization

Both filtering and sampling keep information about a selected subset of locations in the original data space, and drop information about the remaining locations. In contrast, sketch techniques bring together information about every point in the original data. A first approach is to generate the noisy data M', and produce the sketch of this: using the Count Sketch (described in Section 2), we would have significant noise in each bucket in the sketch. We can improve on this considerably by observing that, viewing the sketch as a query, the sketch has low sensitivity. That is, we can view sketching as a mechanism for publishing data similar to histograms: once we fix the hash function h, each bucket contains the sum of counts of a set of individuals, and this forms a partition of the input data. Thus, over d rows of the sketch, the sensitivity is $\Delta_s = d$, and we can achieve privacy by adding the appropriate amount of random noise to each entry in the sketch. Consequently:

LEMMA 4. *The sketch mechanism generates a sketch of size $B \times d$ in time $O((n + B)d)$. The variance of point estimates from the sketch is $O(\|M\|_2/Bd + d/\epsilon^2)$.*

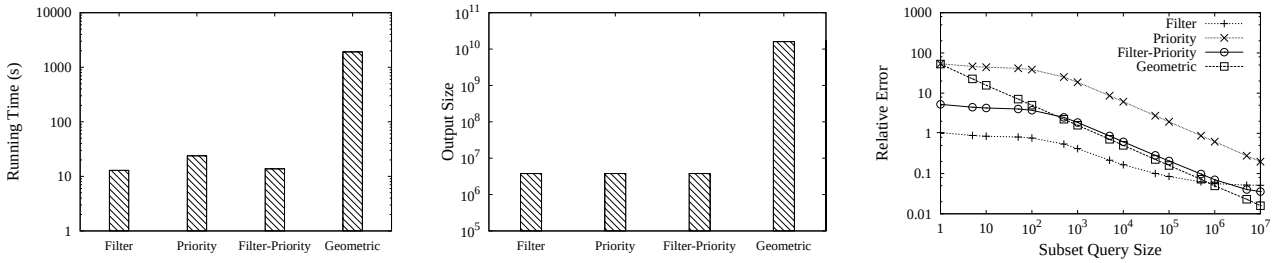
PROOF. The time cost follows from the fact that we have to map each non-zero location in the input to d rows of the sketch, and then add noise to each entry. Since the sketch is typically much smaller than the input domain, there is no need for subsequent sampling or filtering. The variance of each estimate, due to sketching, is proportional to the Euclidean norm of the data scaled by the number of buckets, $\|M\|_2/B$ [2]. The noise added for privacy simply adds to this variance, in the amount of $O(d^2/\epsilon^2)$ for noise with parameter (ϵ/d). Taking the average of d repetitions scales the variance down by a factor of d to $O(\|M\|_2/Bd + d/\epsilon^2)$. □

Thus there is a tradeoff for setting the parameter d: increasing d reduces the sketching error, but increases the privacy error. In an experimental study, we were able to use large values of B, so the second term dominated, meaning that the optimal setting was to pick small values of d, such as $d = 1$. We observed that the error on subset queries was considerably higher than for sampling/filtering, so we omit a detailed study of sketching from this presentation.

5.2 Dyadic Ranges and Wavelets

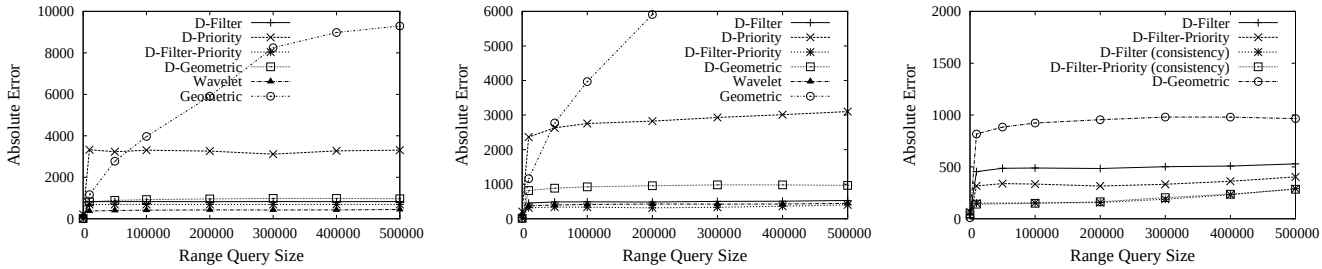
Range queries that touch many entries tend to have much higher error than small queries. Although in expectation the sum of noise values is 0 (so query answers are expected to be correct), its variance is linear in the number of entries touched by the query [7]. In practice, the observed errors tend to be proportional to the standard deviation (i.e., the square root of the number of cells touched).

Dyadic Ranges and Wavelets. A natural way to make range queries more accurate is to publish anonymized data at multiple levels of granularity, so that any range can be decomposed into a small number of probes to the published data. For one-dimensional data, the canonical approach is *dyadic ranges*: Build a (binary) tree over the domain of the data. For each leaf, count the number of data values in the interval corresponding to that leaf. For each internal node u, compute the sum of counts over all the leaves in u's subtree. We can then publish these counts, at all levels, in a privacy-preserving manner. Let h denote the height of this tree. In general, $h = \log m$. Each individual's data now affects h counts, i.e., all the node counts on the path from the individual's leaf to the root. Hence, the sensitivity increases by a factor of h, and the noise in each count is higher. It is well known that any range query can be answered as a sum of at most $2h$ counts (at most two node counts per level). For large enough ranges, the higher noise added to each count is countered by the smaller number of counts touched by it. Since dyadic ranges publish overlapping information (i.e. counts for each node, and all children of each node), it is possible to apply some postpro-



(a) Running time (b) Output size (c) Accuracy

Figure 7: Experimental results using census income data



(a) Dyadic ranges (uniform data) (b) Dyadic ranges (skewed data) (c) Dyadic ranges (skewed data, consistency)

Figure 8: Impact of Dyadic Ranges for query answering

cessing to the counts, to obtain a reduction in error, in time linear in the number of counts, i.e. $O(m)$ [15, 11].

Haar wavelets are quite similar to dyadic ranges in structure. The principal difference is that the wavelet transform is orthonormal, and generates m linearly independent wavelet coefficients. The idea of adding noise in the wavelet domain was advocated by Xiao *et al.* [21]. This approach also has a sensitivity of $\log m$, and answers range queries from $\log m$ counts, but with different constants. In theory and in practice the results for dyadic ranges and wavelets are quite similar: the variance of estimators for range queries under both techniques is proportional to $O(\log^3 m)$.

Combining Transforms with Summarization. These approaches suffer the same limitations as other methods when applied to sparse data: they still output a dataset of size $O(m)$, which is huge compared to the input data. As a result, they also take time linear in m to perform. Thus, when $m \gg n$, we face the same scalability problems. We observe that it is also possible to compose these variants of private data publishing with summarization: The counts computed by dyadic ranges or wavelets on the original data M become the input dataset N for our private summary methods. We then apply the shortcut approaches from the previous sections to compute N''. The key is to observe that both transforms are *sparsity-preserving*: the number of non-zero entries in N is not much bigger than in the original M. This is not the case for other transforms; for example, the Fourier transform of sparse data is always dense. Our main technical result in this setting is as follows:

THEOREM 5. *We can generate summaries of size s with the same distribution as the laborious approach applied to dyadic ranges or Haar wavelets under high-pass filtering and threshold/priority sampling in (expected) time $O(s + n \log(m/n))$.*

PROOF. The approach is quite direct: from M, we produce the (exact) transform into either wavelets or dyadic ranges, then apply our private summary algorithms to this transform. The theorem follows by observing that the total number of non-zero values in the transform of M can be bounded in terms of n and m. We outline the case for dyadic ranges; wavelets are similar.

Consider each non-zero in M as a leaf in a binary tree. The number of non-zeros in the dyadic range transform of M is the number of distinct nodes touched by following the paths from each non-zero to the root. Each non-zero can touch at most one node in any level, giving a crude bound of $n \log m$. This can be tightened by observing that for levels close to the root, there may be fewer than n nodes. More precisely, for the top $\log n$ levels, there are at most n nodes in total. Thus we bound the total number of nodes touched, and hence the number of non-zeros in the transform as:

$$O(n + n(\log m - \log n)) = O(n \log(m/n)).$$

We generate and count all the non-zeros of the transform exactly. This can be done in space $O(\log m)$ by walking over the input in sorted order. This output can be fed into any of the summarization mechanisms from Section 3 directly. We can also compute the exact number of zeros, and choose from these which zeros to upgrade. □

One subtlety with this approach for the wavelet transform is that we produce the output in the transformed space, i.e., we output the noisy set of s wavelet coefficients. This is in contrast to [21], which presents the output in the original (data) domain. The reason is that the s wavelet coefficients do not necessarily lead to sparse output, and may require $O(m)$ space to represent exactly as counts.

This approach extends naturally to multiple dimensions of data. However, care is needed: the sensitivity grows exponentially with the number d of dimensions, as $\log^d m$. As each range query is answered by summing $O(\log^d m)$ counts, there are fewer queries benefiting from dyadic ranges or wavelets, as their sizes must increase rapidly with the dimension.

Consistency Checks for Dyadic Ranges. The work of [15, 11] describes how to perform post-processing on dyadic range counts

to form consistent least-squares estimators. In our setting, we treat this as impractical, due to the high cost of materializing all counts. Instead, we propose a consistency-inspired heuristic for dyadic ranges, via the inherent correlation between counts along the same leaf to root path. For dyadic ranges over the original data, a node count is never smaller than the counts in the node's descendants. Therefore, it is natural to try to enforce a similar condition in the summary data, by modifying some entries after publication.

For filter summaries over dyadic ranges, we impose the following post-processing step. If a node u is selected in the summary M'', but at least one of u's ancestors v is not selected, then we drop u from M''. The intuition for this is straightforward: Since $M(v) \geq M(u)$, v had a higher chance to pass the filter after noise addition. The fact that it did not is strong evidence that its count (and thus u's count) is small, and likely zero. The evidence of v's absence from the summary trumps the evidence of u's presence, so we drop u.

This condition is less meaningful when the summary is a random sample: it may omit some nodes v as part of the random sampling, so the absence of some node does not give a strong reason for dropping its descendants. However it is appropriate to apply the combination of filter and priority sampling described in Section 3.4, if we do so *after* filtering but *before* sampling. This means that we must work on the output of the filtering, which can be large (but still much smaller than m), and so we require more working space than the size of the final summary. However, this may be an acceptable quality/efficiency tradeoff for some data.

5.3 Experiments with Transforms

In this section, we show the result of experiments that combine the dyadic range technique with the filtering and sampling methods to produce a bounded output size, and evaluate the performance on range queries. Specifically, we consider *Filter*, *Priority*, and *Filter-Priority* on dyadic ranges (dubbed *D-Filter, D-Priority and D-Filter-Priority*), as well as the geometric mechanism both with and without dyadic ranges (*Geometric*, resp. *D-Geometric*). We also include the standard *Wavelet* transform, which takes $O(m)$ time and space. Our attempt to combine wavelet with filtering and sampling resulted in considerably degraded query accuracy, so we do not include it. This is because, under the weighting scheme of [21], coefficients which are important for accurate query answering have low magnitude. We leave modifying the scheme to retain important coefficients for future work. We set the number of sampled dyadic ranges to $s = 10^5$, which is equal to the number of non-zeros in the original dataset. We report results for various range query sizes.

Figure 8(a) shows that *D-Priority* is the worst of our private summary techniques on dyadic ranges. This is because the sensitivity of the dyadic ranges is large (i.e., $\log m$), so the noise is also large, giving zero entries among dyadic ranges a high probability to be in the sample. *D-Filter* and *D-Filter-Priority* are much better, since they filter out most of the original zero counts; even if we drop some counts with very small magnitude, this does not dramatically affect the accuracy of the answer on the whole range. We also note that dyadic ranges significantly improve accuracy: *D-Geometric* is far better than *Geometric* for all range query sizes $R \geq 10^4$ (i.e., at least 1% of the domain). Moreover, the accuracy is stable when using dyadic ranges: it depends only very weakly on the range size, since each query probes about the same number of entries (i.e., twice the height of the dyadic range tree).

Figure 8(b) shows the same experiment when the data is skewed, i.e., the non-zeros in the data are more clustered in the domain. This effectively changes the sparsity of the resulting dyadic ranges, which have more zeros in certain tree nodes than others. Now *D-Filter* is better than *D-Geometric* since it can eliminate more noise from these "light" nodes in the dyadic range tree.

Wavelet has good accuracy in both experiments, but is slow and has a large output size of $O(m)$. We computed the running time of all these techniques, while varying the data density from 10^{-1} to 10^{-3}. We observed the same trend as in Figure 4—sampling and filtering are always much faster than both wavelet and the geometric mechanism applied to dyadic ranges. For larger density values, the reduction in running time for our techniques is less pronounced, by a factor of about 2 to 4 times, since all methods incur the extra cost for building the dyadic ranges. The summary size is still orders of magnitude smaller than the unfiltered size of M'.

Overall, we conclude that both dyadic ranges and wavelets are compatible with our private summary framework. Dyadic ranges are highly effective when we anticipate range queries that touch more than a small fraction of the domain. The accuracy of wavelets suffers when combined with our summary methods, and we leave further investigations for future work.

Consistency checks. We also experimented with applying consistency checks to the dyadic ranges, as discussed in Section 5.2. This is done for *Filter*, and for *Filter-Priority* after filtering, but before priority sampling. Note that applying these checks on the geometric mechanism is not helpful, since there are very few entries set to zero that can be used to filter descendant nodes. Figure 8(c) shows that applying consistency checks reduces the errors of our two techniques by 30% to 60%. This confirms that consistency checks help improve accuracy when the data is highly sparse and non-uniform, since they further eliminate noise from originally zero entries. For more uniform datasets, the same trend is present, but is less pronounced: the improvement is closer to 10%.

6. CONCLUDING REMARKS

Differential privacy is a powerful mechanism for releasing data without violating the privacy of the data subjects. In this paper, we have proposed a general framework for computing private summaries of sparse data. Such summaries are an effective way to release data under differential privacy without overwhelming the data user or the data owner. The accuracy of query answering from the released summary compares favorably to the laborious approach of publishing vast data with geometric noise. In many cases, our private summaries even improve query accuracy, by removing a lot of the noise without compromising privacy. Both filtering and sampling are effective in different cases, but the combined *Filter-Priority* method seems generally useful across a wide variety of settings. On data which is sparse, as is the case for most realistic examples, the cost of creating the summary is low, and the benefit only improves as the data dimensionality increases. When we expect that range queries are prevalent, the summaries can be combined with techniques such as dyadic ranges. The benefits of the summary still hold (compact, accurate, fast to compute), and the query accuracy increases even further for most range queries.

7. REFERENCES

[1] 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. *PODS*, 2007.

[2] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *ICALP*, 2002.

[3] B.-C. Chen, D. Kifer, K. LeFevre, and A. Machanavajjhala. *Privacy-Preserving Data Publishing*. NOW publishers, 2009.

[4] G. Cormode and D. Srivastava. Anonymized data: Generation, models, usage. *SIGMOD*, 2009.

[5] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. *SIGCOMM*, 2003.

[6] N. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6), 2007.

[7] C. Dwork. Differential privacy. *ICALP*, pages 1–12, 2006.

[8] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. *SIGMOD* , 2002.

[9] J. Gehrke, D. Kifer, and A. Machanavajjhala. Privacy in data publishing. *IEEE ICDE*, 2010.

[10] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. *STOC*, 2009.

[11] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially-private histograms through consistency. *VLDB*, 2010.

[12] S. Isaacman, R. Becker, R. Cáceres, S. Kobourov, J. Rowland, and A. Varshavsky. A tale of two cities. *HotMobile*, 2010.

[13] D. Kifer. Attacks on privacy and deFinetti's theorem. *SIGMOD*, 2009

[14] D. Kifer and B.-R. Lin. Towards an axiomatization of statistical privacy and utility. *PODS*, 2010.

[15] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. *PODS*, 2010.

[16] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. *IEEE ICDE*, 2008.

[17] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *SIGMOD*, 2009.

[18] F. McSherry and K. Talwar. Mechanism design via differential privacy. *IEEE FOCS*, 2007.

[19] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. NOW publishers, 2005.

[20] F. Olken. *Random Sampling from Databases*. PhD thesis, Berkeley, 1997.

[21] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE ICDE*, 2010.

[22] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. *VLDB SDM Workshop*, 2010.

APPENDIX

In this appendix, we describe the four data sets alluded to in the introduction in greater detail, and show that they are sparse, and have a large domain size.

Census Income Data. Census Income Data has attributes (*Age, Birthplace, Occupation, Income*). Samples can be downloaded from `www.ipums.org`. To represent in a contingency table, we considered income at the various granularities of \$1,000 to \$10,000, and age at granularities of 1 to 5 years. Under various settings, we observed data density at most 5%, and as small as 0.02%.

OnTheMap Data. The US Census Bureau makes available data describing commuting patterns for US residents, as the number of people for each work-home combination (at the census block level), together with other information such as age ranges, salary ranges, and job types. We consider the 47 states available in version 4 of the 2008 data from `http://lehd.did.census.gov/`. We take the location data as the first 2 digits of the census tracts in each county; so each location is identified by "county id + 2-digit tract id". There are 4001 such locations, so the size of the resulting frequency matrix is $m = 1.6 \times 10^7$. The number of non-zeros under this setting is $\sim 8.2 \times 10^5$, so the data density is $\rho = 0.051$. The mean value in each non-zero cell is approximately 150, but with very high variance: many cells have frequency 1 or 2 (and hence should be masked by the addition of noise). The data domain becomes larger and sparser if we include more attributes: adding age, salary and industry, each of which has three values, increases the data size by 27 times while dropping the density to about 0.01.

Adult Data. The Adult Data from the UCI Machine Learning repository, available at `http://archive.ics.uci.edu/ml/datasets/Adult`, has been widely used in data mining, and prior work on privacy [13]. The full data has 14 attributes, but to avoid gridding issues, we projected the data on categorical attributes, i.e, *Workclass, Education, MaritalStatus, Occupation, Relationship, Race, Sex*. This generated data with a density of 0.14%, and an average value in each (non-zero) cell of 9.

Telecom Warehouse Data. AT&T records measurements on the performance of devices in its network in a data warehouse. These measurements include attributes *deviceId, timestamp, Val*, representing a measurement *Val* of each device at a given time stamp. For each day, many gigabytes of data are added to the warehouse. For several natural granularities of the numerical attributes, the observed density ranges from 0.5% to 2%. Therefore, the output of reporting all differentially private counts is 50-200 times larger. Generating, storing and processing data in this form would increase the associated costs by the same factors, making this vastly too expensive. Nevertheless, given the company's data protection policies, and the need for various internal groups to analyze this data, a practical anonymization solution is highly desirable.