Releasing Private Data for Numerical Queries

Yuan Qiu
HKUST
yqiuac@cse.ust.hk

Wei Dong
HKUST
wdongac@cse.ust.hk

Ke Yi
HKUST
yike@cse.ust.hk

Bin Wu
Alibaba Group
binwu.wb@alibaba-inc.com

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

ABSTRACT

Prior work on private data release has only studied counting queries or linear queries, where each tuple in the dataset contributes a value in $[0, 1]$ and a query returns the sum of the values. However, many data analytical tasks involve numerical values that are arbitrary real numbers. In this paper, we present a new mechanism to privatize a dataset D for a given set Q of numerical queries, achieving an error of $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ for each query $w \in Q$, where $\Delta_w(D)$ is the maximum contribution of any tuple in D queried by w. This instance- and query-specific error bound not only is theoretically appealing, but also leads to excellent practical performance.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**.

KEYWORDS

Differential privacy; numerical queries

ACM Reference Format:
Yuan Qiu, Wei Dong, Ke Yi, Bin Wu, and Feifei Li. 2022. Releasing Private Data for Numerical Queries. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539424

1 INTRODUCTION

We study the following fundamental problem in private data release: Let $D \in \mathcal{X}^n$ be a private dataset consisting of n tuples from a universe $\mathcal{X}$. A query is a function $w : \mathcal{X} \to \mathbb{R}$ that assigns a numerical value to each tuple in the universe, and the answer of w on D is defined as $w(D) = \sum_{t \in D} w(t)$. Note that w is defined over the entire universe $\mathcal{X}$ and is public; the query answer on D, namely $w(D)$, is private as it depends on D. Given a set of queries Q, the goal is to release $\tilde{D}$, a differentially private (DP) version of D, such that the answers of all queries in Q on $\tilde{D}$ are as close to those on D as possible.

By appropriately defining w, this problem captures a variety of aggregation queries of interest in private data analysis.

Example 1.1. Consider a dataset of personal incomes, in which each tuple t has an "age" attribute and an "income" attribute, denoted by $t[\text{age}]$ and $t[\text{income}]$ respectively. Then all the following queries are instantiations of the problem above:

(1) The number of people with income between a and b can be obtained by defining
$$w(t) = \mathbf{1}[a \le t[\text{income}] \le b],$$
where $\mathbf{1}[\cdot]$ is the indicator function.

(2) The total (hence the average, by dividing the result by the count obtained from above) income of people whose income is between a and b:
$$w(t) = \mathbf{1}[a \le t[\text{income}] \le b] \cdot t[\text{income}].$$

(3) The following query can be used to compute the variance of income of people in a certain age range:
$$w(t) = \mathbf{1}[a \le t[\text{age}] \le b] \cdot t[\text{income}]^2.$$

(4) The total (or average) weighted income:
$$w(t) = \text{UDF}(t[\text{age}], t[\text{income}]) \cdot t[\text{income}],$$
where $\text{UDF}(\cdot, \cdot)$ is an arbitrary user-defined function that specifies the weight of each person depending on his/her age and income. □

Counting queries, linear queries, and numerical queries. Two special cases of the problem have been studied extensively in the literature [8, 10, 12, 14, 15, 19, 21, 23, 24]: When the codomain of w is $\{0, 1\}$, it is called a *counting query*; more generally, when the codomain is $[0, 1]$, it is called a *linear query*. For a set $\mathcal{L}$ of arbitrary linear queries, the *Private Multiplicative Weights (PMW)* mechanism [10] is the state-of-the-art solution, which guarantees that every query in $\mathcal{L}$ can be answered with error $O(\sqrt{n \log |\mathcal{L}| \log |\mathcal{X}|})$.[1] This error bound is the best achievable (up to polylogarithmic factors) if $\mathcal{L}$ is arbitrary; for queries with special structures (such as range queries, e.g., query (1) in Example 1.1), the error can be further lowered [15, 17, 18].

This paper is concerned with the case where the codomain of w is the entire $\mathbb{R}$, hence called a *numerical query* to differentiate it from the two special cases above. Queries (2), (3), (4) in Example 1.1 are all numerical queries, while (1) is a counting query. Numerical queries are very common, especially when the data itself is numerical. In fact, even if the data is categorical, $w(t)$ can still take numerical values, e.g., $w(t)$ may be the credit score of a customer based his/her categorical attributes.

[1] We suppress the dependency on the privacy parameters in the introduction, which will be spelled out in the technical sections.

For simplicity, for most parts of the paper we consider a discrete, bounded codomain for w, i.e., $w : \mathcal{X} \rightarrow [\Delta] = \{0, 1, \dots, \Delta\}$, for some potentially large integer Δ that is a power of 2 (such as 2^{32} in practice); we show how to deal with the entire real codomain in Section 5.

Normalization. Although prior work has not considered numerical queries explicitly, a standard method to reduce a numerical query w to a linear query is *normalization*: Letting $\Delta_w := \max_{t \in \mathcal{X}} w(t)$, we can normalize w to a linear query $\ell_w(t) := w(t)/\Delta_w$. Then we invoke an existing DP mechanism (e.g., PMW) to compute a privatized $\tilde{D}$ over the set of linear queries $\mathcal{L} = \{\ell_w : w \in Q\}$. Finally, we scale the answer back, i.e., returning $\Delta_w \cdot \ell_w(\tilde{D})$ as an answer for $w(D)$. The error of this method (assuming we use PMW) is thus $\tilde{O}(\sqrt{n} \cdot \Delta_w)$.[2]

However, this error bound can be very large, since Δ_w is the maximum $w(t)$ over the entire universe $\mathcal{X}$, not just the tuples in the dataset D. For instance, in the context of Example 1.1, D is a particular private dataset obtained by some organization, while $\mathcal{X}$ would be all the people in the world; or more formally, it consists of all the valid (age, income) combinations that may appear in any D. For queries with special structures, such as query (2) in Example 1.1, we may have $\Delta_w = b$; in general, however, Δ_w can be as large as Δ. A natural attempt is to normalize with $\Delta_w(D) := \max_{t \in D} w(t)$ taking the maximum $w(t)$ only over D, which would yield an error bound of $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$. Note that this instance-specific bound is much more desirable since $\Delta_w(D) \ll \Delta_w$ on most real-world datasets D. However, normalization with $\Delta_w(D)$ has two issues: (1) $\Delta_w(D)$ is sensitive to D, so using it directly violates DP. (2) Although $\ell_w(t) := w(t)/\Delta_w(D) \in [0, 1]$ for all $t \in D$, it is no longer a linear query, which requires $\ell_w(t) \in [0, 1]$ for all tuples t in the universe $\mathcal{X}$. Fundamentally, all DP mechanisms for linear queries rely on the sensitivity of $\ell_w(D)$ being 1, i.e., changing any tuple in D to an arbitrary other tuple in $\mathcal{X}$ (which may or may not exist in D) changes the value of $\ell_w(D)$ by at most 1. Since the sensitivity of $w(\cdot)$ is Δ_w, we have to normalize with Δ_w in order to bring its sensitivity down to 1.

Truncation. In this paper, we show how to achieve the $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ error bound. Our starting point is the special case where Q consists of just one numerical query w, for which a recently proposed DP mechanism achieves $\tilde{O}(\Delta_w(D))$ error [13]. The idea is very simple: We first find a privatized $\bar{\Delta}_w(D)$ such that $\bar{\Delta}_w(D) \leq \Delta_w(D)$ and only $\tilde{O}(1)$ tuples t in D have $w(t) \geq \bar{\Delta}_w(D)$. This addresses issue (1) above. Then, we truncate the query as $\bar{w}(t) := \min\{w(t), \bar{\Delta}_w(D)\}$. After truncation, we can normalize using $\bar{\Delta}_w(D)$, which guarantees that the normalized query has sensitivity 1, solving issue (2) above. The truncation introduces an error of $\tilde{O}(\Delta_w(D))$ while normalization has error $O(\bar{\Delta}_w(D)) \leq O(\Delta_w(D))$, so the total error is $\tilde{O}(\Delta_w(D))$.

However, extending this idea to multiple queries is nontrivial. There are two straightforward methods, neither of which yields satisfactory error bounds. The first is to use the DP mechanism of [13] to answer each query in Q, after dividing the privacy budget using DP composition theorems. Even with advanced DP composition [7], this method has an error of $\tilde{O}(\sqrt{|Q|} \cdot \Delta_w(D))$. This is undesirable since $|Q|$ can be much (even exponentially) larger than n. In fact, all past work on multiple linear queries aims at reducing the dependency on $|Q|$ from polynomial to logarithmic, as achieved by PMW as well as other mechanisms designed for special classes of linear queries.

The second straightforward adaptation of [13] is to use it only once (thus avoiding the composition problem) to find a global privatized $\bar{\Delta}(D) \leq \Delta(D) := \max_{t \in D, w \in Q} w(t)$. Then we use $\bar{\Delta}(D)$ to truncate all queries in Q, i.e., replace each $w \in Q$ with $\bar{w}(t) := \min\{w(t), \bar{\Delta}(D)\}$. Finally, we normalize all the truncated queries using $\bar{\Delta}(D)$ and invoke PMW. The error of this method is $\tilde{O}(\sqrt{n} \cdot \Delta(D))$. Although this bound has a logarithmic dependency on $|Q|$, it is not necessarily better than the composition-based method, as $\Delta(D)$ can be much larger than $\Delta_w(D)$ for many queries, e.g., query (2) in Example 1.1, or when only a small number of tuples in D are queried by w.

The new mechanism. In this paper, we present a new DP mechanism for numerical queries achieving the $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ error bound. Table 1 compares our result with the three baseline approaches mentioned above. We note that while the three baseline error bounds are incomparable to each other, our error bound dominates all of them as long as $|Q| > n$. The table also lists the three main reasons why the $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ bound is better: it has a logarithmic dependency on $|Q|$, and the $\Delta_w(D)$ term is specific to both the query w and the instance D.

Example 1.2. Consider all the range-income queries formed by query (2) in Example 1.1. Then $|Q| = \Theta(|\mathcal{X}|^2) \geq \Omega(n^2)$, so clearly $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ is much smaller than $\tilde{O}(\sqrt{|Q|} \cdot \Delta_w(D))$. For a query w with income range $[a, b]$, we have $\Delta_w = b$; $\Delta(D)$ = highest income of any person in the dataset D; and $\Delta_w(D)$ = highest income of any person in D whose income is in the range $[a, b]$. Clearly, we have $\Delta_w(D) < \Delta_w < \Delta(D)$.

Next consider all range-variance queries formed by query (3) in Example 1.1. For these queries, we have $\Delta_w = \Delta$ = highest income (squared) of any person in the world, while $\Delta_w(D)$ is the highest income of any person in D aged between a and b. Now we have $\Delta_w(D) < \Delta(D) < \Delta_w$. These two examples illustrate that, while there is no clear winner between $\Delta(D)$ and Δ_w, $\Delta_w(D)$ is always better, since it takes the advantage of both the specific D (the actual incomes in D can be much smaller than Δ) and the specific w (in an actual D, smaller ages tend to correspond to smaller incomes). □

To summarize, we make the the following contributions in this paper:

(1) We initiate the study of private data release for numerical queries, which generalize counting queries and linear queries, and are important for many private data analysis tasks involving numerical data.
(2) We put forward three baseline solutions for the problem by adapting prior methods, and point out their deficiencies.
(3) We design new DP mechanisms for the problem with an error bound of $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$ for an arbitrary set Q of numerical queries, which dominates all the three baseline methods.
(4) We show how to further improve the mechanism, in terms of both running time and the error bound, for a *decomposable* set of queries.

[2]The $\tilde{O}$ notation suppresses polylogarithmic factors.

Mechanism	Error Bound for $w \in Q$	Many Queries?	Query-Specific?	Instance-Specific?
Normalization	$\tilde{O}(\sqrt{n} \cdot \Delta_w)$	✓	✓	×
Composition	$\tilde{O}(\sqrt{\lvert Q \rvert} \cdot \Delta_w(D))$	×	✓	✓
Global truncation	$\tilde{O}(\sqrt{n} \cdot \Delta(D))$	✓	×	✓
New method	$\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$	✓	✓	✓

Table 1: Comparison of error bounds.

(5) Through an extensive experimental study, we demonstrate that our methods are not only theoretically superior, but also perform very well in practice.

2 PRELIMINARIES

Two datasets $D, D' \in \mathcal{X}^n$ are neighboring instances if they differ by one tuple.

Definition 2.1 (Differential Privacy [7]). A mechanism $\mathcal{M} : \mathcal{X}^n \to \mathcal{O}$ satisfies (ε, δ)-differential privacy if for any pair of neighboring datasets $D \sim D'$ and any subset of outputs $O \subset \mathcal{O}$,

$$\Pr[\mathcal{M}(D) \in O] \le e^{\varepsilon} \cdot \Pr[\mathcal{M}(D') \in O] + \delta\,. \quad (1)$$

It is well known that adding noise calibrated to the (global) sensitivity of a query protects differential privacy.

Lemma 2.2. *Given a function $f : \mathcal{X}^n \to \mathbb{R}$, a mechanism that outputs $\mathcal{M}(D) = f(D) + \mathrm{Lap}(\mathrm{GS}/\varepsilon)$ satisfies $(\varepsilon, 0)$-DP, where $\mathrm{GS} = \max_{D,D' \in \mathcal{X}^n : D \sim D'} |f(D) - f(D')|$ is the sensitivity of f.*

We will need the following useful properties of differential privacy:

Lemma 2.3 (Composition Theorem [7]). *If $\mathcal{M}$ is an adaptive composition of differentially private mechanisms $\mathcal{M}_1, \dots, \mathcal{M}_T$, where each $\mathcal{M}_i$ satisfies $(\varepsilon_0, 0)$-DP, then $\mathcal{M}$ satisfies $(\varepsilon(\varepsilon_0, \delta, T), \delta)$-DP, where*

(1) $\varepsilon(\varepsilon_0, \delta, T) = T\varepsilon_0$ *for* $\delta = 0$*;*

(2) $\varepsilon(\varepsilon_0, \delta, T) = \varepsilon_0\sqrt{2T \log \frac{1}{\delta}} + T\varepsilon_0(e^{\varepsilon_0} - 1)$ *for any* $\delta > 0$.

Lemma 2.4 (Group Privacy [7]). *If $\mathcal{M}$ is an $(\varepsilon_0, 0)$-DP mechanism, then for any two datasets D, D' that differ by at most k tuples, $\mathcal{M}$ satisfies* (1) *with $\varepsilon = k\varepsilon_0$ and $\delta = 0$.*

We will use the PMW mechanism as an important building block:

Lemma 2.5 (PMW [10]). *For any set $\mathcal{L}$ of linear queries over a universe $\mathcal{X}$ and every $\varepsilon_0 > 0$ and an integer $T \ge 1$, there exists a mechanism $\mathcal{M}$ such that for any dataset $D \in \mathcal{X}^n$, with probability $1 - \beta$, all queries $\ell \in \mathcal{L}$ can be answered on $\tilde{D} = \mathcal{M}(D)$ within error at most*

$$\alpha(\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|) = O\left(n\sqrt{\frac{\log|\mathcal{X}|}{T}} + \frac{\log(|\mathcal{L}|/\beta)}{\varepsilon_0}\right).$$

The mechanism runs in T rounds, with each round being $(\varepsilon_0, 0)$-DP and taking $\tilde{O}(|\mathcal{X}| \cdot |\mathcal{L}|)$ time.

The explicit error bounds of the PMW mechanism follow by combining Lemma 2.3 and 2.5: After running for T rounds, PMW satisfies $(\varepsilon(\varepsilon_0, \delta, T), \delta)$-DP by Lemma 2.3. For this to be (ε, δ)-DP with $\delta > 0$, one may set $T = \tilde{\Theta}(\varepsilon n)$ and $\varepsilon_0 = \Theta(\varepsilon/\sqrt{T \ln(1/\delta)})$ to achieve an optimal error bound

$$\alpha(\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|) = O\left(\sqrt{\frac{n \cdot \log(|\mathcal{L}|/\beta) \cdot \sqrt{\log|\mathcal{X}| \log(1/\delta)}}{\varepsilon}}\right).$$

For $\delta = 0$, the best choice is $T = \tilde{\Theta}((\varepsilon n)^{2/3})$ and $\varepsilon_0 = \varepsilon/T$, which yields

$$\alpha(\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|) = O\left(\left(\frac{n^2 \cdot \log(|\mathcal{L}|/\beta) \cdot \log|\mathcal{X}|}{\varepsilon}\right)^{1/3}\right).$$

Nevertheless, it has been observed that setting T to a constant tends to work well in practice [10]. Note that α also depends on δ, β, but we omit them in the notation as they will be fixed in the later development, while we will invoke PMW with different $\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|$.

3 ARBITRARY NUMERICAL QUERIES

In this section, we describe our mechanism for an arbitrary set Q of numerical queries. To achieve the query- and instance-specific error bound proportional to $\Delta_w(D)$, we need to find a truncation threshold $\bar{\Delta}_w(D)$ for each w. However, we cannot afford to do so for each $w \in Q$ as in the composition-based baseline method. Our idea is to find all these truncation thresholds in a "batching" mode by employing PMW. Specifically, our new algorithm consists of the following steps. First, we generate a set of counting queries $C(Q)$ from Q, which contain enough information for us to find an approximate truncation threshold for every query in Q. These counting queries will then be answered by PMW. From the answers on $C(Q)$, we extract the truncation thresholds $\bar{\Delta}_w(D)$ for all $w \in Q$. Finally, we truncate and normalize each w using $\bar{\Delta}_w(D)$, converting them to a set of linear queries $\mathcal{L}(Q)$, which are answered by another PMW instance, before scaling the answers back. For conceptual simplicity, this outline uses two separate PMW instances; later we show that they can actually be combined into one.

3.1 Finding Truncation Thresholds

We will find, in a DP fashion, a truncation threshold $\bar{\Delta}_w(D)$ for each w such that (1) $\bar{\Delta}_w(D)$ is not too much larger than $\Delta_w(D) := \max_{t \in D} w(t)$, and (2) not too many tuples $t \in D$ have $w(t) > \bar{\Delta}_w(D)$. The first condition bounds the error due to normalization while the second condition bounds the truncation error. The following lemma summarizes these two conditions more quantitatively.

Lemma 3.1. *Let $\alpha(\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|)$ be defined as in Lemma 2.5. For any T, δ, there exists an $(\varepsilon(\varepsilon_0, \delta, T), \delta)$-DP mechanism that, given a dataset D and numerical queries Q, returns a $\bar{\Delta}_w(D)$ for every $w \in Q$,* *such that with probability* $1-\beta$*, we have* $\bar{\Delta}_w(D) \le 2\Delta_w(D)$ *and*

$$\sum_{t \in D} \mathbf{1}[w(t) > \bar{\Delta}_w(D)] \le 2\alpha(\varepsilon_0, n, |\mathcal{X}|, (1+\log\Delta)|Q|),$$

for all $w \in Q$*.*

We now describe the mechanism. Given a set Q of numerical queries, we define the following set of counting queries

$$C(Q) := \{c_{w,\tau} : w \in Q, \tau \in \{0, 1, 2, 4, \ldots, \Delta/2\}\},$$

where

$$c_{w,\tau}(t) = \mathbf{1}[w(t) > \tau].$$

Note that $|C| = (1+\log\Delta)|Q|$. Then we invoke PMW to return a $\tilde{D}$ that can be used to answer all counting queries in C within error $\alpha := \alpha(\varepsilon_0, n, |\mathcal{X}|, |C|)$. Next, for each $w \in Q$, we perform a doubling search to find the largest τ such that $c_{w,\tau}(\tilde{D}) \le \alpha$. Algorithm 1 describes the process more precisely. Below we prove Lemma 3.1.

Algorithm 1: Finding Approximate Maximum Weight

```
   Input: Numerical queries $Q$, dataset $D$, universe $\mathcal{X}$
   Input: Privacy budget $(\varepsilon, \delta)$
   Output: $\bar{\Delta}_w(D)$ for every $w \in Q$
1  $C \leftarrow \{c_{w,\tau} : w \in Q, \tau \in \{0, 1, 2, 4, \ldots, \Delta/2\}\}$;
2  $\tilde{D}, \alpha \leftarrow \mathrm{PMW}_{\varepsilon,\delta}(\mathcal{X}, D, C)$;
3  for $w \in Q$ do
4      if $c_{w,0}(\tilde{D}) \le \alpha$ then
5          Output $\bar{\Delta}_w(D) = 0$;
6      else
7          $\tau \leftarrow 1$;
8          while $\tau < \Delta$ do
9              if $c_{w,\tau}(\tilde{D}) \le \alpha$ then
10                 Output $\bar{\Delta}_w(D) = \tau$;
11             else
12                 $\tau \leftarrow 2\tau$;
13             end
14         end
15         Output $\bar{\Delta}_w(D) = \Delta$;
16     end
17 end
```

Proof. Privacy is straightforward: Note that the only access to the dataset D happens at line 2. The remaining steps do not consume privacy by the post-processing property of DP. The whole mechanism thus satisfies (ε, δ)-DP.

We next prove the two utility guarantees in the lemma, conditioned on the event that all queries in C can be answered on $\tilde{D}$ with error at most α, which happens with probability $1-\beta$.

Consider any $w \in Q$. We first show that $\sum_{t \in D} \mathbf{1}[w(t) > \bar{\Delta}_w(D)] \le 2\alpha$. If the mechanism outputs $\bar{\Delta}_w(D) = \Delta$ at line 15, then $\sum_{t \in D} \mathbf{1}[w(t) > \Delta] = 0$ as Δ is an upper bound on $w(t)$. Otherwise, it must be that $c_{w,\tau}(\tilde{D}) \le \alpha$ for the output $\tau = \bar{\Delta}_w(D)$. By the accuracy guarantee of $\tilde{D}$, we know $c_{w,\tau}(D) - \alpha \le c_{w,\tau}(\tilde{D}) \le \alpha$, which gives $c_{w,\tau}(D) = \sum_{t \in D} \mathbf{1}[w(t) > \tau] \le 2\alpha$.

We now prove that $\bar{\Delta}_w(D) \le 2\Delta_w(D)$. When 0 is returned at line 5, $0 \le 2\Delta_w(D)$ is trivial. Otherwise, let $\tau' = \lfloor \bar{\Delta}_w(D)/2 \rfloor$ be the previous value checked before returning $\bar{\Delta}_w(D)$. We must have $\alpha < c_{w,\tau'}(\tilde{D}) \le c_{w,\tau'}(D) + \alpha$, otherwise τ' would have been the output. This gives $c_{w,\tau'}(D) = \sum_{t \in D} \mathbf{1}[w(t) > \tau'] > 0$, so we have $\tau' < \Delta_w(D) = \max_{t \in D} w(t)$. When $\bar{\Delta}_w(D) = 1$ and $\tau' = 0$, this gives $\Delta_w(D) \ge 1$ so that $\bar{\Delta}_w(D) < 2\Delta_w(D)$. Otherwise, $\bar{\Delta}_w(D) = 2\tau' < 2\Delta_w(D)$. □

The running time of Algorithm 1 is dominated by that of PMW. After that, we issue at most $O(|Q|\log\Delta)$ counting queries on $\tilde{D}$.

3.2 Truncation and Normalization

After finding the truncation thresholds $\bar{\Delta}_w(D)$, we truncate and normalize each numerical query $w \in Q$ into a linear query $\ell_w(t) := \min\{w(t)/\bar{\Delta}_w(D), 1\}$. Then we invoke PMW again on the set of linear queries $\mathcal{L}(Q) := \{\ell_w : w \in Q\}$, and scale the answers back by $\bar{\Delta}_w(D)$.

The error consists of two parts. First, running PMW on $\mathcal{L}(Q)$ has an error of $\alpha(\varepsilon_0, n, |\mathcal{X}|, |Q|)$, which translates into an error of $\bar{\Delta}_w(D) \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, |Q|) = O(\Delta_w(D) \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, |Q|))$ by the first condition in Lemma 3.1. The truncation introduces an error of at most $2\Delta_w(D) \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, (1+\log\Delta)|Q|)$ by the second condition in Lemma 3.1. This yields the following result on arbitrary numerical queries.

Theorem 3.2. *Let* $\alpha(\varepsilon_0, n, |\mathcal{X}|, |\mathcal{L}|)$ *be defined as in Lemma 2.5. For an arbitrary set* Q *of numerical queries, and any* T, δ*, there exists an* $(\varepsilon(\varepsilon_0, \delta, T), \delta)$*-DP mechanism* $\mathcal{M}$ *that, for any dataset* $D \in \mathcal{X}^n$*, answers all queries in* Q *within error* $O(\Delta_w(D) \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, |Q|\log\Delta))$ *with probability* $1-\beta$*. The running time of* $\mathcal{M}$ *is dominated by that of PMW.*

Optimality. Plugging in Lemma 2.3, 2.5, and ignoring polylogarithmic factors, the error bound from Theorem 3.2 is $\tilde{O}(\Delta_w(D) \cdot \sqrt{n})$. Below we argue that this is the best one can hope for. For a single numerical query w, $\Delta_w(D)$ is the downward local sensitivity of w on D [2, 20], which is an instance-specific lower bound in a small local neighborhood. Thus, an error bound of $\tilde{O}(\Delta_w(D))$ can be considered *instance-optimal*, and such instance-optimal DP mechanisms have been developed for a single numerical query [2, 13]. Theorem 3.2 falls short of instance-optimality by an extra $\tilde{O}(\sqrt{n})$ factor, which can be considered as the *optimality ratio*. However, an $\tilde{O}(\sqrt{n})$ optimality ratio is unavoidable by the following simple argument: Suppose there was a DP mechanism that achieves an error of $\tilde{o}(\Delta_w(D) \cdot \sqrt{n})$. Since counting queries are special numerical queries where $\Delta_w(D) = 1$ for all w, this would also imply a DP mechanism for counting queries with error $\tilde{o}(\sqrt{n})$, contradicting the known lower bound [11].

3.3 Combining the Two PMW Instances

The algorithm described above used two PMW instances. Running two PMW instances consumes not only more time, but also more privacy (we need to split the privacy budget to the two instances). We observe that the input to the two PMW mechanisms is the same dataset D, although the queries are different: the first PMW instance runs on the set of counting queries $C(Q)$ while the second on the set of linear queries $\mathcal{L}(Q)$. We cannot combine the two instances directly, since the truncation and normalisation in $\mathcal{L}(Q)$ depends on the $\bar{\Delta}_w(D)$'s, which can only be obtained after running the first PMW instance.

The idea to get around this dependency issue is to simply include in $\mathcal{L}(Q)$ all linear queries using all possible truncation thresholds, i.e., we redefine

$$\mathcal{L}(Q) := \{\ell_{w,\tau} : w \in Q, \tau \in \{1, 2, 4, \ldots, \Delta\},$$

where $\ell_{w,\tau}(t) := \min\{w(t)/\tau, 1\}$. This increases the size of $\mathcal{L}$ to be equal to that of C, both of which are now $(1 + \log\Delta)|Q|$. Nevertheless, since α depends on the number of queries only logarithmically, the impact on the error is negligible. In fact, the increased error is completely offset by the benefit of using all the privacy budget on a single PMW instance. We formalize the full algorithm in Algorithm 2, which maintains the error guarantee in Theorem 3.2.

Algorithm 2: Mechanism for Numerical Queries

```
Input: Numerical queries $Q$, dataset $D$, universe $\mathcal{X}$
Input: Privacy budget $(\varepsilon, \delta)$
Output: Privatized answers $\tilde{w}(D)$ for every $w \in Q$
1  $C \leftarrow \{c_{w,\tau} := \mathbf{1}[w(t) > \tau] : w \in Q, \tau \in \{0,1,2,4,\ldots,\Delta/2\}\}$;
2  $\mathcal{L} \leftarrow \{\ell_{w,\tau} := \min\{w(t)/\tau, 1\} : w \in Q, \tau \in \{1,2,4,\ldots,\Delta\}\}$;
3  $\tilde{D}, \alpha \leftarrow \mathrm{PMW}_{\varepsilon,\delta}(\mathcal{X}, D, C \cup \mathcal{L})$;
4  for $w \in Q$ do
5      if $c_{w,0}(\tilde{D}) \le \alpha$ then
6          Output $\tilde{w}(D) = 0$;
7      else
8          $\tau \leftarrow 1$;
9          while $\tau < \Delta_w$ do
10             if $c_{w,\tau}(\tilde{D}) \le \alpha$ then
11                 Output $\tilde{w}(D) = \tau \cdot \ell_{w,\tau}(\tilde{D})$;
12             else
13                 $\tau \leftarrow 2\tau$;
14             end
15         end
16         Output $\tilde{w}(D) = \Delta_w \cdot \ell_{w,\Delta_w}(\tilde{D})$;
17     end
18 end
```

4 DECOMPOSABLE QUERIES

The mechanism from the previous section works for a set of queries that may define arbitrary mappings $w : \mathcal{X} \to [\Delta]$. It achieves an error of $\tilde{O}(\Delta_w(D) \cdot \sqrt{n})$ while taking time $\tilde{O}(T \cdot |\mathcal{X}| \cdot |Q|)$, neither of which can be improved if we aim for arbitrary numerical queries. In this section, we show how to improve both the error and running time for a set of decomposable queries, which form an important class of numerical queries commonly encountered in practice.

Definition 4.1. A set of queries Q is said to be *decomposable* if there exists an equivalence relation R over $\mathcal{X}$ and a function $g : \mathcal{X} \to [\Delta]$, such that every $w \in Q$ can be written as

$$w(t) = f_w([t]_R) \cdot g(t), \tag{2}$$

for some $f_w : \mathcal{X}/R \to [0, 1]$.

Here $[t]_R \in \mathcal{X}/R$ denotes the equivalent class induced by R that contains tuple t. Note that while $g(\cdot)$ is common to the entire Q, $f_w(\cdot)$ can be different for each $w \in Q$.

By definition, any set of numerical queries Q is trivially decomposable, simply by using the identity equivalence relation $R_0 = \{(t, t) : t \in \mathcal{X}\}$. As $\mathcal{X}/R_0 = \mathcal{X}$, we can set $g(\cdot) \equiv \Delta$ and $f_w([t]_R) = w(t)/\Delta$ to satisfy Definition 4.1. We are more interested in decompositions where (a) $|\mathcal{X}/R| \ll |\mathcal{X}|$, and/or (b) the $f_w(\cdot)$'s for all $w \in Q$ are counting queries with a common structural property. We will show in this section how (a) reduces the running time and (b) leads to improved error bounds.

Example 4.2. Consider the four (sets of) queries in Example 1.1. Query (1) satisfies (b) using the identity R_0 and $g(\cdot) \equiv 1$, while the $f_w(\cdot)$'s are all range queries. Query (2) satisfies (b) using the identity R_0 and $g(t) = t[\text{income}]$, while the $f_w(\cdot)$'s are also range queries. Query (3) satisfies both (a) and (b), with R putting all tuples of the same age into an equivalence class (so that $\mathcal{X}/R = \mathbf{dom}(\text{age})$) and $g(t) = t[\text{income}]^2$, while the $f_w(\cdot)$'s are range queries on the age attribute. Query (4) is only trivially decomposable if there is no restriction on UDF$(\cdot,\cdot)$. □

4.1 Reducing Universe Size

The running time of PMW is proportional to $|\mathcal{X}|$, the universe size. The issue can be particularly bad for numerical queries as $|\mathcal{X}|$ is usually at least Δ. Consider query (3) in Example 1.1. Suppose the domain of age is $[1, 128]$ and the domain of income (squared) is $[1, \Delta = 2^{32}]$. Then we have $|\mathcal{X}| = 2^{40}$, which renders it impossible to run PMW. Below we show how to reduce the effective universe for PMW to $|\hat{\mathcal{X}}| = (1 + \log\Delta)|\mathcal{X}/R|$, at the cost of increasing the error bound by a $\log(\Delta)$ factor. For query (3) in Example 1.1, we have $\mathcal{X}/R = \mathbf{dom}(\text{age})$, so the reduced universe has size $|\hat{\mathcal{X}}| = 4224$, and PMW can now run in a reasonable amount of time.

The reduction works as follows. We define a new universe $\hat{\mathcal{X}} = \mathcal{X}/R \times \{1, 2, 4, \ldots, \Delta\}$. Clearly $|\hat{\mathcal{X}}| = (1 + \log\Delta)|\mathcal{X}/R|$. An instance $D \in \mathcal{X}^n$ is transformed to $\hat{D} \in \hat{\mathcal{X}}^{(1+\log\Delta)n}$ as follows. For each $t \in D$, we write $g(t)$ in binary representation: $g(t) = \sum_{i=0}^{\log\Delta} g_i(t)2^i$ where $g_i(t) \in \{0, 1\}$. For each $i = 0, 1, \ldots, \log\Delta$, we place a tuple $([t]_R, g(i)2^i)$ in $\hat{D}$. Formally,

$$\hat{D} = \uplus_{t\in D} \uplus_{i=0}^{\log\Delta} \{([t]_R, g(i)2^i)\},$$

where $\uplus$ denotes multiset union.

For any query $w \in Q$, we also define a transformed query $\hat{w}(x, y) := f_w(x) \cdot y$ on the transformed dataset $\hat{D}$. It can be verified that

$$\hat{w}(\hat{D}) = \sum_{t\in D} \sum_{i=0}^{\log\Delta} f_w([t]_R)g(i)2^i = \sum_{t\in D} f_w([t]_R)g(t) = w(D).$$

Finally, we run Algorithm 2 on $\hat{D}$ over the universe $\hat{\mathcal{X}}$ to answer queries $\{\hat{w}\}$, which provide answers to $w \in Q$. The running time is now $\tilde{O}(T \cdot |\hat{\mathcal{X}}| \cdot |Q|) = \tilde{O}(T \cdot |\mathcal{X}/R| \cdot |Q|)$.

On the other hand, since each original tuple $t \in D$ generates $(1 + \log\Delta)$ tuples in $\hat{D}$. To ensure privacy for the original dataset D, we need to apply group privacy (Lemma 2.4) and use $\hat{\varepsilon}_0 = \varepsilon_0/(1 + \log\Delta)$ for each round of the PMW mechanism. Then following a similar analysis as for Theorem 3.2, we obtain the following result:

THEOREM 4.3. *For any set of numerical queries Q that is decomposable by equivalence relation R, and any T, δ, there is an $(\varepsilon(\varepsilon_0/(1+\log\Delta), \delta, T), \delta)$-DP mechanism that runs in $\tilde{O}(T \cdot |\mathcal{X}/R| \cdot |Q|)$ time, and with probability $1-\beta$, it answers every $w \in Q$ within error $O(\Delta_w(D) \cdot \alpha(\varepsilon_0/\log\Delta, n\log\Delta, |\mathcal{X}/R|, |Q|\log\Delta))$.*

Remark. Compared with Theorem 3.2, Theorem 4.3 has an extra $O(\log\Delta)$ factor to both ε_0 and n. However, since they are in two separate terms in the error bound α (see Lemma 2.5), we have only one $O(\log\Delta)$ factor increase in the total error. Meanwhile, since the universe size has been reduced to $|\mathcal{X}/R|$, this offsets the increase in error to some extent.

4.2 Queries with Structural Properties

For an arbitrary set of linear queries, the PMW mechanism achieves the optimal error $\tilde{O}(\sqrt{n})$. For counting queries with certain structural properties, there are mechanisms achieving better error bounds. Specifically, it is known that for a set of orthogonal range counting queries in constant dimensions, the error can be reduced to $\alpha = \tilde{O}(1/\varepsilon)$ [6]; for a set of half-space counting queries in d dimensions, the optimal error is $\alpha = \tilde{O}(n^{\frac{1}{2}-\frac{1}{2d}}/\varepsilon)$ [17]. More generally, the optimal error bound depends on the hereditary discrepancy of the query set, and there are DP mechanisms matching this optimal error up to polylogarithmic factors [15, 18]. However, existing work only supports counting queries. Although not explicitly stated, they can be relatively easily modified to support "weighted" counting queries, where each tuple t is assigned a weight $g(t) \in [0, 1]$, and the query asks for the total weight of all tuples t in the query. This is the same as decomposable numerical queries, except that the weights $g : \mathcal{X} \to [\Delta]$ have a much larger codomain. This means that directly using normalization would yield an error of $O(\Delta \cdot \alpha)$. Below we show how to reduce it to $\tilde{O}(\Delta_w(D) \cdot \alpha)$.

Let $c_{w,\tau}(t) = \mathbf{1}[w(t) > \tau]$ and $\ell_{w,\tau}(t) = \min\{w(t)/\tau, 1\}$ for all $w \in Q$ and $\tau = 0, 1, 2, 4, \ldots, \Delta$, as in Section 3. Recall that Algorithm 2 relies on accurately answering these counting and linear queries. We consider each in turn.

Answering $\{c_{w,\tau}\}$. First consider the counting queries $\{c_{w,\tau}\}$. For a set Q of numerical queries that can be decomposed as in (2) such that each $f_w([t]_R)$ is a counting query, we can rewrite $c_{w,\tau}$ as

$$c_{w,\tau}(t) = f_w([t]_R) \cdot \mathbf{1}[g(t) > \tau].$$

Instead of using PMW to answer $\{c_{w,\tau}\}$, which can only achieve error $\alpha = \tilde{O}(\sqrt{n})$, we will use a DP mechanism that exploits the structural properties of $\{f_w\}$. To do so, we partition the universe $\mathcal{X}$ into $1 + \log\Delta$ subsets: $\mathcal{X}_i = \{t \in \mathcal{X} : 2^{i-1} < g(t) \le 2^i\}$ for $i = 1, \ldots, \log\Delta$ and $\mathcal{X}_0 = \{t \in \mathcal{X} : 0 < g(t) \le 1\}$.[3] Correspondingly, the given dataset $D \in \mathcal{X}^n$ is also partitioned to $D = \uplus_{i=0}^{\log\Delta} D_i$, where $D_i = \{t \in D : t \in \mathcal{X}_i\}$. By parallel composition [7], we can afford to run an (ε, δ)-DP mechanism on each D_i, and the composed mechanism on D still satisfies (ε, δ)-DP.

For each D_i, we answer all the counting queries $\{f_w\}$ on D_i using a DP mechanism tailored designed for $\{f_w\}$ with error bound α. Note that the query result of f_w on D_i is $f_w(D_i) = \sum_{t\in D_i} f_w([t]_R)$.

[3]We may assume have $g(t) > 0$ for all $t \in \mathcal{X}$. Otherwise such t can be removed from $\mathcal{X}$ without affecting any query.

Observe that for any τ that is a power of 2, we have

$$\begin{aligned} c_{w,\tau}(D) &= \sum_{t\in D} f_w([t]_R) \cdot \mathbf{1}[g(t) > \tau] \\ &= \sum_{i=1+\log\tau}^{\log\Delta} \sum_{t\in D_i} f_w([t]_R) \\ &= \sum_{i=1+\log\tau}^{\log\Delta} f_w(D_i). \end{aligned}$$

Thus, each counting query $c_{w,\tau}(D)$ is the sum of $O(\log\Delta)$ counting queries $f_w(D_i)$, each of which can be answered with error α. So $c_{w,\tau}(D)$ can be answered within error $O(\alpha\log\Delta)$. Put into the framework of Section 3, this means that the truncation error is $O(\Delta_w(D) \cdot \alpha\log\Delta)$.

Answering $\{\ell_{w,\tau}\}$. Next, consider the linear queries $\{\ell_{w,\tau}\}$. We rewrite $\ell_{w,\tau}(D)$ as

$$\begin{aligned} \ell_{w,\tau}(D) &= \sum_{t\in D} \min\{w(t)/\tau, 1\} \\ &= \sum_{i\le\log\tau} w(D_i)/\tau + \sum_{i>\log\tau} \sum_{t\in D_i} f_w([t]_R) \\ &= \sum_{i\le\log\tau} w(D_i)/\tau + \sum_{i>\log\tau} f_w(D_i). \end{aligned}$$

The $f_w(D_i)$'s are already available, each with error α, so it only remains to find $w(D_i)$. We rewrite it as

$$w(D_i) = \sum_{t\in D_i} f_w([t]_R) g(t) = 2^i \sum_{t\in D_i} f_w([t]_R) g(t)/2^i.$$

Note that $\sum_{t\in D_i} f_w([t]_R) g(t)/2^i$ is a weighted counting query where tuple t has weight $g(t)/2^i$. Since all tuples $t \in D_i$ have $g(t) \le 2^i$, all these weights are in $[0, 1]$, and we can invoke an existing DP mechanism to answer all these weighted counting queries with error α, which translates to an error of $2^i\alpha$ for each $w(D_i)$. So, the total error for the first term of $\ell_{w,\tau}(D)$ is bounded by $\sum_{i\le\log\tau} 2^i\alpha/\tau = O(\alpha)$, which is dominated by the total error $O(\alpha\log\Delta)$ in the second term. Put into the framework of Section 3, this means that the normalization error is also $O(\Delta_w(D) \cdot \alpha\log\Delta)$.

Finally, note that unlike Algorithm 2, which invokes only one PMW instance, here we need to invoke an existing DP mechanism for $\{f_w\}$ twice: the first is for the counting queries and the second is weighted counting. Thus, we need to split the privacy budget. Putting everything together, we conclude with the following result.

THEOREM 4.4. *Given any set of numerical queries Q that is decomposable by equivalence relation R such that $w(t) = f_w([t]_R)g(t)$ where $\{f_w\}$ are counting queries that can be answered by an (ε, δ)-DP mechanism within error α, there is an $(2\varepsilon, 2\delta)$-DP mechanism that answers every $w \in Q$ with error $O(\Delta_w(D) \cdot \alpha\log\Delta)$.*

Remark. When Q satisfies both condition (a) and (b), Theorem 4.4 is preferable to Theorem 4.3. This is because the α in Theorem 4.4 is always smaller than that in Theorem 4.3 when the queries have certain structural properties. Furthermore, the running times of existing DP mechanisms for range queries [6] and half-space queries [17] depend on the universe size only logarithmically, so the benefit of a reduced universe size is insignificant.

5 DEALING WITH THE REAL CODOMAIN

In this section, we describe techniques to deal with the case $w : \mathcal{X} \to \mathbb{R}$. Firstly, we can separate the positive and negative numerical values, by splitting each query $w \in Q$ into $w^+(t) = \max\{w(t), 0\}$ and $w^-(t) = \max\{-w(t), 0\}$. Note that $w(t) = w^+(t) - w^-(t)$, so we can answer any $w \in Q$ by returning $w^+(D) - w^-(D)$. The error is only scaled up by a factor of 2. Now both $w^+(t)$ and $w^-(t)$ are non-negative, so it suffices to consider the case $w : \mathcal{X} \to \{0\} \cup \mathbb{R}^+$.

Next, we consider fractional values. In fact, the only place where we rely on integer values for the general mechanism in Section 3 is in the doubling search in Algorithm 2. In particular, when $c_{w,0}(\tilde{D}) > \alpha$, we know that 0 cannot be used as a truncation threshold, and we jumped to $\tau = 1$. This is because for integer values, any truncation threshold $\tau \in (0, 1)$ will function exactly the same as $\tau = 0$. However, when $w(t)$ can take fractional values, this no longer holds: Suppose $\Delta_w(D) < 1$, and we use 1 as the truncation threshold. The truncation error is not affected, as using a larger threshold leads to less value getting truncated. Yet, the normalization error will be proportional to 1 instead of $\Delta_w(D)$, resulting in an error of $O(\max\{\Delta_w(D), 1\} \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, |Q| \log \Delta))$, which can be higher than the desired error guarantee $O(\Delta_w(D) \cdot \alpha(\varepsilon_0, n, |\mathcal{X}|, |Q| \log \Delta))$ if $\Delta_w(D) \ll 1$. To handle this very small $\Delta_w(D)$, we modify line 8 of Algorithm 2 as $\tau \leftarrow b$ for a parameter $b \in (0, 1]$. There are now $(1 + \log(\Delta/b))$ counting queries for each $w \in Q$, and the total error changes to

$$O\left(\max\{\Delta_w(D), b\} \cdot \alpha\left(\varepsilon_0, n, |\mathcal{X}|, |Q| \log \frac{\Delta}{b}\right)\right).$$

Since α depends on the number of queries logarithmically, the increase from $|Q|$ to $|Q| \log \frac{\Delta}{b}$ has a negligible impact on the error, which means that we can set b conservatively to a very small value so that $b < \Delta_w(D)$, hence not affecting our error guarantee asymptotically.

Our DP mechanism in Section 4.1 also relies on $g(t)$ being an integer, since we decompose it into $(1 + \log \Delta)$ bits. If $g(t)$ is a real number, we can simply also decompose the fractional part of $g(t)$, up to $\log(1/b)$ bits after the decimal point. Correspondingly, we need to replace every occurrence of $(1 + \log \Delta)$ to $(1 + \log \frac{\Delta}{b})$.

Finally, we reduce the codomain to a bounded discrete domain $w : \mathcal{X} \to [\bar{\Delta}]$. Using the private-radius mechanism [3], we can find a privatized $\bar{\Delta} \le \max\{b, 2\Delta(D)\}$ such that only $\tilde{O}(1)$ tuples in D have $w(t) > \bar{\Delta}$. Now we do a global truncation by redefining each w as $\bar{w} := \min\{w(t), \bar{\Delta}\}$, which causes at most $\tilde{O}(\max\{b, \Delta_w(D)\})$ error to each w. Again using a conservatively small b, this additional error is negligible.

6 EXPERIMENTAL EVALUATION

We have evaluated our mechanisms on various datasets against the three baseline methods, using the datasets from [10]. While [10] only considered counting queries, we study numerical queries. The Adult [4] dataset, also referred to as the Census Income dataset, contains 48,842 records over 14 attributes. The Transfusion [4, 22] dataset contains 748 records over 4 attributes. We also included another Bank Marketing [4, 16] dataset, which contains 41,445 records over 16 attributes.

We mainly tested the pure-DP setting, i.e., $\delta = 0$; approximate-DP (i.e., $\delta > 0$) would result in similar improvements to all methods. For each set of queries, we compare the following mechanisms.

- The normalization mechanism, which normalizes every $w \in Q$ by Δ_w, and answers them with the PMW mechanism.
- The composition mechanism, which answers each $w \in Q$ using the mechanism in [13] with $(\varepsilon/|Q|, 0)$ privacy budget for each query.
- The global truncation mechanism, which first truncates every query by $\bar{\Delta}(D)$, and then applies the normalization mechanism.
- Our general mechanism (Algorithm 2) for arbitrary numerical queries in Section 3, which uses PMW mechanism as a subroutine. We always use the universe reduction technique from Section 4.1 when applicable.
- For 1D range numerical queries, we also compare with the mechanism in Section 4.2. We find that in two or higher dimensions, the mechanism in Section 4.2 is not as practically competitive as our general mechanism, despite its better theoretical error bound.

For the PMW mechanism, we follow implementation in [10]. In particular, we set $T = 10$ as recommended. Note that in Algorithm 2, we require PMW to also report α, the maximum error of all queries. While this can be directly calculated using the error upper bound, the bound turns out to be too loose. Therefore, after PMW returns $\tilde{D}$, we compute the actual maximum error $\alpha = \max_{\ell \in \mathcal{L}} |\ell(D) - \ell(\tilde{D})|$. Note that α is a sensitive value involving D, so we cannot use α directly. However, its sensitivity is 1 as on any two neighboring datasets, we have $|\ell(D) - \ell(D')| \le 1$ for any linear query ℓ. So by sparing a small constant fraction of ε, we can use privatized version of α with only $O(1/\varepsilon)$ noise (Lemma 2.2).

In each set of experiments, we present the errors of all queries in Q using a box plot, which shows the maximum, median, and 25%/75% percentiles of the $|Q|$ errors.

6.1 Results on the Adult Dataset

We used 4 attributes from the dataset: a person's age (age), education level (edu), capital loss (loss), and capital gain (gain). The queries are designed as follows. We first consider a set of rectangle ranges on $\mathbf{dom}(\text{age}) \times \mathbf{dom}(\text{edu})$, while aggregating on loss. We set the domains of the attributes as $\mathbf{dom}(\text{age}) = [1, 100]$ and $\mathbf{dom}(\text{edu}) = [1, 16]$. Each query selects a closed interval of ages whose length is between 0 and 10, and one or two consecutive education levels. There are 1045 intervals on age and 31 different conditions on education level, leading to a total of $|Q| = 32395$ queries. Note that for this set of queries, $\Delta_w = \Delta$ for all $w \in Q$.

The maximum value on the loss attribute is 4396. In [10], they simply set $\mathbf{dom}(\text{loss}) = [0, \Delta = 4396]$. However, this setting actually violates privacy, since the maximum loss is a sensitive value. Nevertheless, we still follow this setting, while noting that for strict privacy, Δ should be set to the largest possible loss that is independent of the actual dataset, e.g., 2^{32}, which would render the baseline methods extremely bad. Thus, our experimental comparison actually gives the largest possible advantage to the baseline methods.

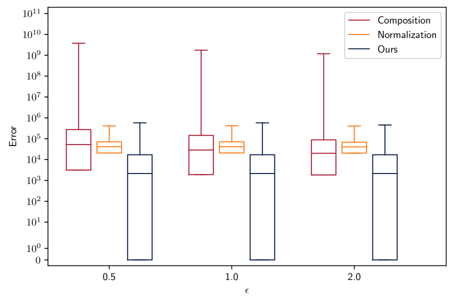

Figure 1: Adult, Aggregate "loss"

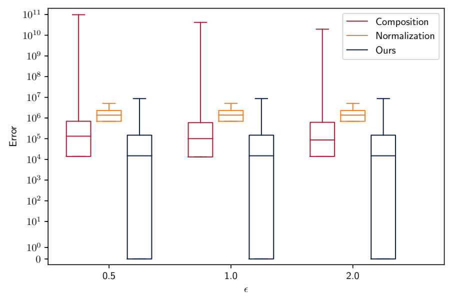

Figure 2: Adult, Aggregate "gain"

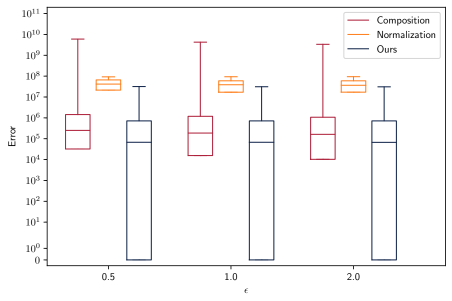

Figure 3: Bank

With this (non-private) setting of Δ, we have $\Delta(D) = \Delta$, so the normalization mechanism coincides with the global truncation mechanism. The whole universe $\mathcal{X} = \textbf{dom}(\text{age}) \times \textbf{dom}(\text{edu}) \times \textbf{dom}(\text{loss})$ has size 7×10^6, and we use the universe reduction technique in Section 4.1 to reduce it to $|\mathcal{X}'| = 100 \times 16 \times (1 + \log \Delta) = 20,800$.

Figure 1 shows the results on varying ε. Note that the errors are shown in log-scale. The composition mechanism has the worst performance in terms of both maximum error and median error. This is because the privacy budget ε has to be shared by all the $|Q|$ queries. For most queries, our mechanism significantly outperforms the two baseline methods, especially for queries with smaller ranges. This is precisely due to our both query- and instance-specific error bound $\tilde{O}(\sqrt{n} \cdot \Delta_w(D))$. On the other hand, the error box of the normalization mechanism is much narrower, since its error bound $\tilde{O}(\sqrt{n} \cdot \Delta_w) = \tilde{O}(\sqrt{n} \cdot \Delta)$ is the same for all queries.

Nevertheless, our mechanism has a slightly worse maximum error compared with the normalization mechanism. This is actually because the normalization mechanism is given the tightest, private-breaching $\Delta_w = \Delta = \max_{w \in Q} \Delta_w(D)$, so the worst query $w \in Q$ does not benefit from our mechanism, which selects query-specific truncation thresholds. If a larger Δ (which respects more privacy) were used, the errors of all queries for the normalization mechanism would grow proportionally.

We also tested the same queries but replacing the aggregation attribute by "gain". This time, we set $\textbf{dom}(\text{gain}) = [0, \Delta = 99999]$ as opposed to using the maximum gain in D. The results are shown in Figure 2. As expected, the normalization mechanism becomes much worse in this case.

6.2 Results on the Bank Dataset

For the Bank dataset, we used numerical attributes for a client's age (age), average yearly balance (balance), and Boolean attributes for whether or not the person has housing loan (housing) or personal loan (loan). Each query still selects an interval of ages whose length is between 0 to 10, with all possible conditions on "housing" and "loan", and finally aggregates on "balance". This leads to $|Q| = 4180$ numerical queries. The difference from the Adult dataset is that the "balance" attribute contains both positive and negative values. We assume $\textbf{dom}(\text{balance}) = [-2^{13}, 2^{17}]$, which is just enough to include all values appearing in the dataset. We use the technique from Section 5 to separate each query into two to deal with positive and negative values respectively.

Figure 3 shows the results. The composition mechanism is slightly better in this case, since there are relatively a small number of queries. Yet, its maximum error is still very large. In terms of median error, composition and our method are both much better than the normalization mechanism (or the global truncation mechanism). This is because this dataset is highly skewed: A small number of people have very large balances, but they are not queried by most queries. The normalization technique thus suffers from this skewness as it normalizes all queries using the same Δ. On the other hand, the composition mechanism and our new mechanism are able to handle this skewness very nicely.

6.3 Results on the Transfusion Dataset

On the Blood Transfusion dataset, we consider two attributes: the number of blood donations a person has made (frequency) and number of months since his/her first donation (time). The domains are $\textbf{dom}(\text{frequency}) = [1, 50]$ and $\textbf{dom}(\text{time}) = [1, 200]$. The universe therefore has size $|\mathcal{X}| = 10000$. We are interested in the average duration between two blood donations, which can be calculated as $t[\text{time}]/t[\text{frequency}]$ for a person t in the dataset. To create multiple queries, we apply selection conditions on the "frequency" domain by considering all sub-intervals of $[1, 50]$, which creates 1275 different queries. Each query is written as

$$w(D) = \sum_{t \in D} \mathbf{1}[a \le t[\text{frequency}] \le b] \cdot t[\text{time}]/t[\text{frequency}] .$$

For this set of queries, Δ_w is different for each w, which is $\Delta_w = 200/a$. Therefore the normalization mechanism and the global truncation mechanism differ. As the universe size is moderate, we can directly run the PMW mechanism on the original instance without applying the universe reduction technique which sacrifices accuracy for time.

Figure 4 shows the results varying ε. The composition mechanism is still bad, as there are more queries than the instance size $n = 748$. For this set of queries, the normalization mechanism is generally better-performing than the global truncation one, except for its maximum error. This is because the normalization mechanism is able to take the advantage of a smaller Δ_w for certain queries. For example, for a query that selects only people who donated more than 10 times, its Δ_w is only 20. Our mechanism is slightly better than the normalization mechanism in this case for the same reason that the difference between the maximum value in the dataset vs.

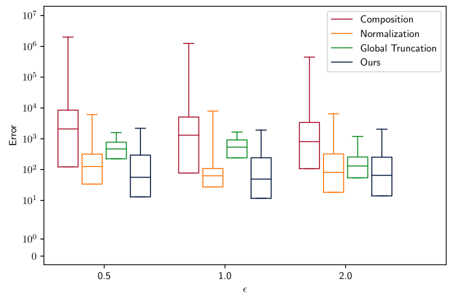

Figure 4: Transfusion, Varying ε

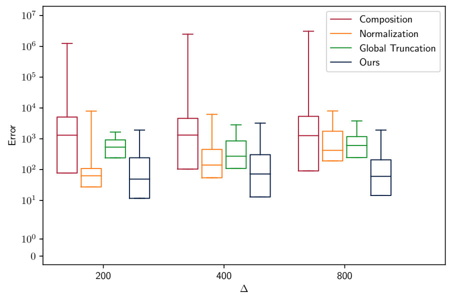

Figure 5: Transfusion, Varying Δ

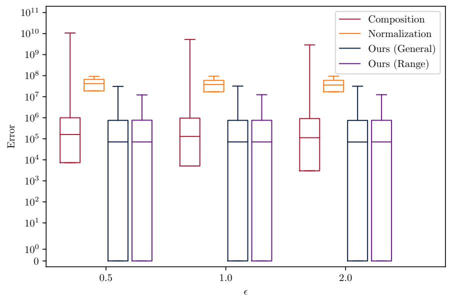

Figure 6: Range Mechanism on Bank

the global sensitivity is small for many queries. It has the smallest median error over all mechanisms.

In Figure 5, we fix $\varepsilon = 1.0$, and change the domain of the attribute "time" to $[1, 400]$ and $[1, 800]$ respectively. Correspondingly, the universe size $|\mathcal{X}|$ is increased, as well as the Δ_w of each query. The instance-independent normalization mechanism is affected the most, while all the other mechanism are instance-specific, thus is robust with respect to the change in the universe size. Our mechanism shows significant improvements when the global sensitivity of queries are large.

6.4 Evaluation on Structured Queries

In previous results, we implemented our general mechanism which does not take advantage of the structural properties of queries. In this section, we implement the mechanism in Section 4.2, using 1D range queries as example. We use queries on the Bank dataset for evaluation. For simplicity, we only consider positive balances this time. The basic weighted range counting mechanism we use is the same as [1, 5], which builds a synopsis using dyadic intervals [9].

Results are shown in Figure 6. The performance of the range mechanism in practice is similar to that of the general mechanism, with a smaller maximum error. But there are two advantages. First, the running time of the mechanism is much smaller than that of the general mechanism. Second, the range mechanism indeed supports all range conditions on the "age" attribute with no additional costs, while the general mechanism uses the number of queries as a parameter.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under grants 16201318, 16201819, and 16205420, and by Alibaba Group through Alibaba Innovative Research Program.

REFERENCES

[1] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Trans. Inf. Syst. Secur.* 14, 3 (2011), 26:1–26:24.
[2] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *Proc. International Conference on Management of Data (SIGMOD)*. ACM.
[3] Wei Dong and Ke Yi. 2021. Universal Private Estimators. *CoRR* abs/2111.02598 (2021).
[4] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[5] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *Proc. Symposium on Theory of Computing (STOC)*. ACM, 715–724.
[6] Cynthia Dwork, Moni Naor, Omer Reingold, and Guy N. Rothblum. 2015. Pure Differential Privacy for Rectangle Queries via Private Partitions. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, Vol. 9453. 735–751.
[7] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.
[8] Alexander Edmonds, Aleksandar Nikolov, and Jonathan R. Ullman. 2020. The power of factorization mechanisms in local and central differential privacy. In *Proc. Symposium on Theory of Computing (STOC)*. ACM, 425–438.
[9] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. 2002. How to Summarize the Universe: Dynamic Maintenance of Quantiles. In *Proc. International Conference on Very Large Data Bases (VLDB)*. 454–465.
[10] Moritz Hardt, Katrina Ligett, and Frank McSherry. 2012. A Simple and Practical Algorithm for Differentially Private Data Release. In *Conference on Neural Information Processing Systems (NeurIPS)*.
[11] Moritz Hardt and Kunal Talwar. 2010. On the geometry of differential privacy. In *Proc. Symposium on Theory of Computing (STOC)*. ACM, 705–714.
[12] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *Proc. VLDB Endow.* 3, 1 (2010), 1021–1032.
[13] Ziyue Huang, Yuting Liang, and Ke Yi. 2021. Instance-optimal Mean Estimation Under Differential Privacy. In *Conference on Neural Information Processing Systems (NeurIPS)*.
[14] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau, and Andrew McGregor. 2010. Optimizing linear counting queries under differential privacy. In *Proc. Symposium on Principles of Database Systems (PODS)*. ACM, 123–134.
[15] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *VLDB J.* 24, 6 (2015), 757–781.
[16] Sérgio Moro, Paulo Cortez, and Paulo Rita. 2014. A data-driven approach to predict the success of bank telemarketing. *Decis. Support Syst.* 62 (2014), 22–31.
[17] S. Muthukrishnan and Aleksandar Nikolov. 2012. Optimal private halfspace counting via discrepancy. In *Proc. Symposium on Theory of Computing Conference (STOC)*. ACM, 1285–1292.
[18] Aleksandar Nikolov, Kunal Talwar, and Li Zhang. 2013. The geometry of differential privacy: the sparse and approximate cases. In *Proc. Symposium on Theory of Computing Conference (STOC)*. ACM, 351–360.
[19] Vibhor Rastogi and Suman Nath. 2010. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proc. International Conference on Management of Data (SIGMOD)*. ACM, 735–746.
[20] Salil P. Vadhan. 2017. The Complexity of Differential Privacy. In *Tutorials on the Foundations of Cryptography*. Springer, 347–450.
[21] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential privacy via wavelet transforms. In *Proc. International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 225–236.
[22] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. 2009. Knowledge discovery on RFM model using Bernoulli sequence. *Expert Syst. Appl.* 36 (2009), 5866–5871.
[23] Ganzhao Yuan, Yin Yang, Zhenjie Zhang, and Zhifeng Hao. 2016. Convex Optimization for Linear Query Processing under Approximate Differential Privacy. In *Proc. International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 2005–2014.
[24] Ganzhao Yuan, Zhenjie Zhang, Marianne Winslett, Xiaokui Xiao, Yin Yang, and Zhifeng Hao. 2015. Optimizing Batch Linear Queries under Exact and Approximate Differential Privacy. *ACM Trans. Database Syst.* 40, 2 (2015), 11:1–11:47.

A EXPERIMENTS ON THE TPC-H BENCHMARK

To test the performance of our mechanism on a larger dataset, we also run experiments on a 1GB TPC-H dataset. We consider the following query modified from TPC-H Q6.

```
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate between '[a]' and '[b]'
    and l_discount between '[c]'-0.01 and '[c]'+0.01
    and l_quantity = '[d]'
```

This query involves 4 attributes from the largest table lineitem, which contains 4 million tuples. The domain of each attribute is obtained from the source code which generates the dataset. For example, `l_extendedprice` can be as large as 105000 and `l_discount` ranges from 0% to 10%.

The query set we consider is specified as follows. Let [a] and [b] iterate through all possible range conditions on shipdate that contains at most 12 months. This creates 942 intervals. Let [c] take any value between 2% and 9% as in the standard specification, and [d] take any value between 1 and 10. There are 75360 queries by considering all the combinations of [a] to [d] as above. The global sensitivity of any query is $\Delta = 105000 * 10\% = 10500$.

The Cartesian product of related attributes form a universe of size around 5 billion. So we apply the decomposition mechanism in Section 4.1. For this query, the decomposition works on the conceptual column that equals

$$g(t) = t[\text{l\_extendedprice}] \cdot t[\text{l\_discount}]$$

for each tuple t.

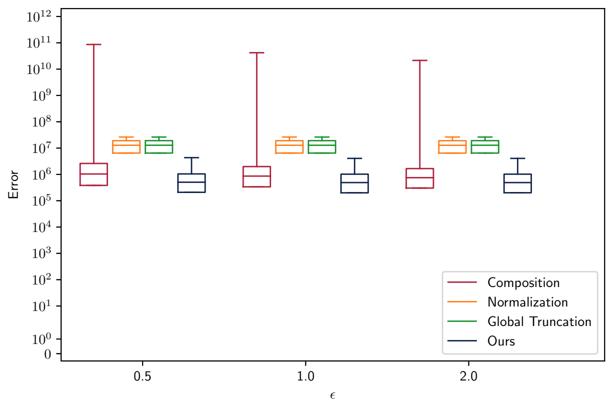

Figure 7: TPC-H Q6

Figure 7 shows the results. The instance is huge while the number of queries is relatively small. This causes the composition mechanism to perform better than the Normalization or Global Truncation mechanism. But its maximum error is still very large. Our mechanism is still the best performing one, despite the fact that theoretically n is much larger than $|Q|$.