A Data- and Workload-Aware Algorithm for Range Queries Under Differential Privacy

Chao Li[†], Michael Hay[‡], Gerome Miklau[†], Yue Wang[†]

[†]University of Massachusetts Amherst
School of Computer Science
{chaoli,miklau,yuewang}@cs.umass.edu

[‡]Colgate University
Department of Computer Science
mhay@colgate.edu

ABSTRACT

We describe a new algorithm for answering a given set of range queries under ϵ-differential privacy which often achieves substantially lower error than competing methods. Our algorithm satisfies differential privacy by adding noise that is adapted to the input data *and* to the given query set. We first privately learn a partitioning of the domain into buckets that suit the input data well. Then we privately estimate counts for each bucket, doing so in a manner well-suited for the given query set. Since the performance of the algorithm depends on the input database, we evaluate it on a wide range of real datasets, showing that we can achieve the benefits of data-dependence on both "easy" and "hard" databases.

1. INTRODUCTION

Differential privacy [8, 9] has received growing attention in the research community because it offers both an intuitively appealing and mathematically precise guarantee of privacy. In this paper we study batch (or non-interactive) query answering of range queries under ϵ-differential privacy. The batch of queries, which we call the *workload*, is given as input and the goal of research in this area is to devise differentially private mechanisms that offer the lowest error for any fixed setting of ϵ. The particular emphasis of this work is to achieve high accuracy for a wide range of possible input databases.

Existing approaches for batch query answering broadly fall into two categories: *data-independent* mechanisms and *data-dependent* mechanisms. Data-independent mechanisms achieve the privacy condition by adding noise that is independent of the input database. The Laplace mechanism is an example of a data-independent mechanism. Regardless of the input database, the same Laplacian noise distribution is used to answer a query. More advanced data-independent mechanisms exploit properties of the workload to achieve greater accuracy, but the noise distribution (and therefore the error) remains fixed for *all* input databases.

Data-dependent mechanisms add noise that is customized to properties of the input database, producing different error rates on different input databases. In some cases, this can result in significantly lower error than data-independent approaches. These mechanisms typically need to use a portion of the privacy budget to learn about the data or the quality of a current estimate of the data. They then use the remaining privacy budget to privately answer the desired queries. In most cases, these approaches do not exploit workload.

A comparison of state-of-the-art mechanisms in each category reveals that each has advantages, depending on the "hardness" of the input database. If the database is viewed as a histogram, databases with large uniform regions can be exploited by these algorithms, allowing the data-dependent mechanisms to outperform data-independent ones. But on more complex datasets, e.g. those with many regions of density, data-dependent mechanisms break down.

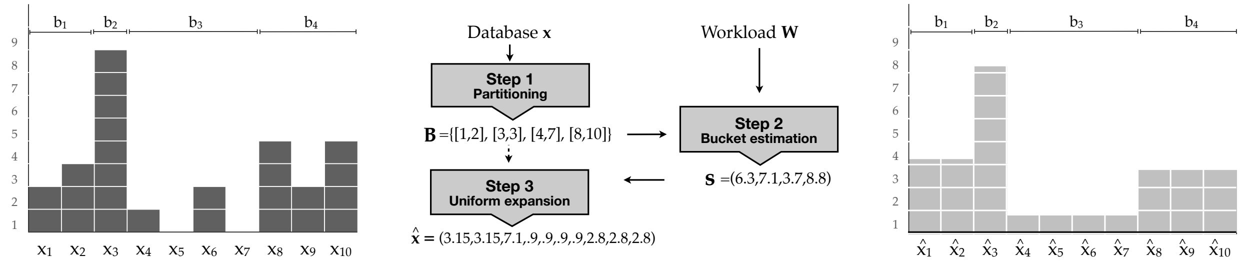
Consider as an example a workload of random range queries and a dataset derived from an IP-level network trace. A state-of-the-art data-dependent mechanism, *Multiplicative Weights and Exponential Mechanism* (MWEM) [12], can achieve 60.12 average per-query error when $\epsilon = 0.1$. For the same ϵ, one of the best data-independent mechanisms for this workload, *Privelet* [20], offers per-query error of 196.6, a factor of 3.27 worse. But other datasets have properties that are difficult to exploit. On a dataset based on the HEP-PH citation network, MWEM has average per-query error of 722.3 with $\epsilon = 0.1$, while the error of the data-independent mechanism is still 196.6 for this workload, a factor of 3.67 better.

Such a large variation in the relative performance of mechanisms across data sets is a major limitation of current approaches. This is especially true because it is typically necessary to select a mechanism without seeing the data.

Contributions. First, we propose a novel 2-stage mechanism for answering range queries under ϵ-differential privacy. On inputs where existing data-dependent mechanisms do well, our mechanism achieves lower error by a factor of up to 6.86 compared with the state of the art. On inputs where existing data-dependent mechanisms do poorly, our mechanism achieves error comparable to state-of-art data-independent mechanisms. Second, we present an efficient algorithm in the first stage that partitions the domain into uniform regions. Compared with other differentially private partitioning algorithms, our algorithm generates much better partitions and runs in time that is only quasilinear in the size of the domain. Third, we design a new, efficient algorithm in the second stage that computes scaling factors for a hierarchical set of range queries. Unlike existing hierarchical strategies, our method allows a non-uniform budget distribution across queries at the same level, which leads to a strategy that is more finely tuned to the workload, and thus more accurate.

To our knowledge, our mechanism is the first data-aware mechanism that provides significant improvement on databases with easy-to-exploit properties yet does not break-down on databases with complex distributions.

Proceedings of the VLDB Endowment, Vol. 7, No. 5
Copyright 2014 VLDB Endowment 2150-8097/14/01.


b1
b2
b3
b4
9
8
7
6
5
4
3
2
1
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
Database x
Step 1
Partitioning
B ={[1,2], [3,3], [4,7], [8,10]}
Workload W
Step 2
Bucket estimation
s =(6.3,7.1,3.7,8.8)
Step 3
Uniform expansion
x̂ = (3.15,3.15,7.1,.9,.9,.9,.9,2.8,2.8,2.8)
x̂1 x̂2 x̂3 x̂4 x̂5 x̂6 x̂7 x̂8 x̂9 x̂10

(a) True database $\mathbf{x}$

(b) Algorithm flow chart

(c) Private output $\hat{\mathbf{x}}$

Figure 1: Overview and example execution for the DAWA mechanism.

Algorithm Overview

We give an overview to our new mechanism and an example below.

The *Data-Aware/Workload-Aware* (DAWA) mechanism is an ϵ-differentially-private algorithm that takes as input a workload of range queries, $\mathbf{W}$, and a database, $\mathbf{x}$, represented as a vector of counts. The output is an estimate $\hat{\mathbf{x}}$ of $\mathbf{x}$, where the noise added to achieve privacy is adapted to the input data and to the workload. The DAWA algorithm consists of the following three steps, the first two of which require private interactions with the database. To ensure that the overall algorithm satisfies ϵ-differential privacy, we split the total ϵ budget into ϵ_1, ϵ_2 such that $\epsilon_1 + \epsilon_2 = \epsilon$ and use these two portions of the budget on the respective stages of the algorithm.

Step 1: Private Partitioning

The first step selects a partition of the domain that fits the input database. We describe (in Sec. 3) a novel differentially private algorithm that uses ϵ_1 budget to select a partition such that within each partition bucket, the dataset is approximately *uniform*. This notion of uniformity is later formalized as a cost function but the basic intuition is that if a region is uniform, then there is no benefit in using a limited privacy budget to ask queries at a finer granularity than these regions—the signal is too small to overcome the noise. The output of this step is B, a partition of $\mathbf{x}$ into k buckets, *without* counts for the buckets.

Step 2: Private Bucket Count Estimation

Given the partition B, the second step derives noisy estimates of the bucket counts. Rather than simply adding Laplace noise to the bucket counts, we use a workload-aware method. Conceptually, we re-express the workload over the new domain defined by the partition B, with the buckets in the partition taking the place of $\mathbf{x}$. Then we have a well-studied problem of selecting unbiased measurements (i.e. linear functions of the bucket counts) in a manner that is optimized for the workload. This problem has received considerable attention in past work [6, 7, 15, 16, 23, 24]. We use the basic framework of the matrix mechanism [15], but we propose a new algorithm (described in Sec. 4) for efficiently approximating the optimal measurements for the workload.

Given the selected measurements, we then use the ϵ_2 privacy budget and Laplace noise to privately answer the measurement queries, followed by least-squares inference to derive the output of this step, a noisy estimate $\mathbf{s}$ for the buckets in B.

Step 3: Uniform Expansion

In the last step we derive an estimate for the n components of $\mathbf{x}$ from the k components of the histogram $(B, \mathbf{s})$. This is done by assuming uniformity: the count s_i for each bucket b_i is spread uniformly amongst each position of $\mathbf{x}$ that is contained in b_i. The result is the estimate $\hat{\mathbf{x}}$ for $\mathbf{x}$. Strictly speaking, any range query can be computed from $\hat{\mathbf{x}}$, but the noise is tuned to provide accuracy for precisely the queries in the workload.

The following example illustrates a sample execution of DAWA.

EXAMPLE 1. *For* $n = 10$*, Fig. 1 shows graphically a sample data vector* $\mathbf{x} = (2, 3, 8, 1, 0, 2, 0, 4, 2, 4)$*. A possible output of Step 1 is* $B = \{b_1, b_2, b_3, b_4\}$ *where* $b_1 = [1, 2]$*,* $b_2 = [3, 3]$*,* $b_3 = [4, 7]$*, and* $b_4 = [8, 10]$*. This need not be the optimal partition, as defined in Sec. 3, because the partition selection is randomized. For the sample database* $\mathbf{x}$ *in the figure, the true bucket counts for the partition would be* $(5, 8, 3, 10)$*. The result from Step 2 is a set of noisy bucket counts,* $\mathbf{s} = (6.3, 7.1, 3.6, 8.4)$*. Step 3 then constructs* $\hat{\mathbf{x}}$ *by assuming a uniform distribution for values within each bucket. As it is shown graphically in Fig. 1(c), the final output is*

$$\hat{\mathbf{x}} = (3.15, 3.15, 7.1, .9, .9, .9, .9, 2.8, 2.8, 2.8).$$

The novelty of our approach consists of splitting the overall private estimation problem into two phases: Step 1, which is data-dependent, and Step 2, which is workload-aware. Our main technical contributions are an effective and efficient private solution to the optimization problem underlying Step 1, and an effective and efficient solution to the optimization problem underlying Step 2. We also extend our methods to two-dimensional workloads using spatial decomposition techniques.

A number of recently-proposed methods [3,6,21,22] share commonalities with one or more parts of our mechanism (as described in Sec. 6). But each omits or simplifies an important step and/or they use sub-optimal methods for solving related subproblems. In Sec. 5, an extensive experimental evaluation shows that for workloads of 1- and 2-dimensional range queries, the DAWA algorithm achieves lower error than all competitors on nearly every database and setting of ϵ tested, often by a significant margin.

The paper is organized as follows. We review notation and privacy definitions in Sec. 2. The partitioning algorithm is presented in Sec. 3, and the bucket count estimating algorithm is included in Sec. 4. We extensively compare DAWA with state-of-the-art competing mechanisms in Sec. 5. Related work is discussed in Sec. 6. We conclude and mention future directions in Sec. 7.

2. BACKGROUND

In this section we review notation, basic privacy definitions, and standard privacy mechanisms used throughout the paper.

2.1 Databases and Queries

The query workloads we consider consist of counting queries over a single relation. Let the database I be an instance of a single-relation schema $R(\mathbb{A})$, with attributes $\mathbb{A} = \{A_1, A_2, \ldots, A_k\}$ each having an ordered domain. In order to express our queries, we first transform the instance I into a *data vector* $\mathbf{x}$ consisting of n non-negative integral counts. We restrict our attention to the one- or two-dimensional case. In one dimension, we isolate a single attribute, A_i, and define $\mathbf{x}$ to consist of one coefficient for each element in the domain, $dom(A_i)$. In other words, x_j reports the number of tuples in database instance I that take on the j^{th} value in the ordered domain of A_i. In the two-dimensional case, for attributes A_i, A_j, $\mathbf{x}$ contains a count for each element in $dom(A_i) \times dom(A_j)$. For simplicity, we describe our methods in the one-dimensional case, extending to two-dimensions in Sec. 5.4.

A query workload $\mathbf{W}$ defined on $\mathbf{x}$ is a set of range queries $\{w_1 \dots w_m\}$ where each w_i is described by an interval $[j_1, j_2]$ for $1 \leq j_1 \leq j_2 \leq n$. The evaluation of $w_i = [j_1, j_2]$ on $\mathbf{x}$ is written $w_i(\mathbf{x})$ and defined as $\sum_{j=j_1}^{j_2} x_j$. We use $\mathbf{W}(\mathbf{x})$ to denote the vector of all workload query answers $\langle w_1(\mathbf{x}) \dots w_m(\mathbf{x}) \rangle$.

A histogram on $\mathbf{x}$ is a partition of $[1, n]$ into non-overlapping intervals, called buckets, along with a summary statistic for each bucket. We denote a histogram by $(B, \mathbf{s})$ with B a set of buckets $B = \{b_1 \dots b_k\}$ and $\mathbf{s}$ a set of corresponding statistics $\mathbf{s} = s_1 \dots s_k$. Each b_i is described by an interval $[j_1, j_2]$ and the set of intervals covers $[1, n]$ and all intervals are disjoint. We define the length $|b_i|$ of bucket b_i to be $j_2 - j_1 + 1$.

We associate a summary statistic with each of the k buckets in a histogram. One way to do this is to treat the bucket intervals as range queries and evaluate them on $\mathbf{x}$. We denote this true statistic for bucket b_i by $b_i(\mathbf{x})$ and we use $B(\mathbf{x})$ to denote the vector for true bucket counts. In other cases, the summary statistics are noisy estimates of $B(\mathbf{x})$, denoted $\mathbf{s} = s_1 \dots s_k$.

Throughout the paper we use the *uniform expansion* of a histogram $(B, \mathbf{s})$. It is a data vector of length n derived from B by assuming uniformity for counts that fall within bucket ranges.

DEFINITION 1 (UNIFORM EXPANSION). *Let expand be a function that takes a histogram $H = (B, \mathbf{s})$ with buckets $B = \{b_1 \dots b_k\}$ and statistics $\mathbf{s} = s_1 \dots s_k$, and uniformly expands it. Thus, $expand(B, \mathbf{s})$ is an n-length vector $\mathbf{y}$ defined as:*

$$y_j = \frac{s_{t(j)}}{|b_{t(j)}|}$$

where $t(j)$ is the function that maps position j to the index of the unique bucket in B that contains position j for $j \in [1, n]$.

In our algorithms, both the choice of a histogram and the value of the histogram statistics have the potential to leak sensitive information about $\mathbf{x}$. Both must be computed by a differentially private algorithm. Suppose that a differentially private algorithm returns histogram $H = (B, \mathbf{s})$ where the statistics have noise added for privacy. We use $\hat{\mathbf{x}}$ to denote the uniform expansion of H, i.e., $\hat{\mathbf{x}} = expand(B, \mathbf{s})$. Since the vector $\hat{\mathbf{x}}$ is a differentially private estimate for $\mathbf{x}$, we can use it to answer any query w as $w(\hat{\mathbf{x}})$.

We are interested in how accurately $\hat{\mathbf{x}}$ approximates $\mathbf{x}$. The *absolute error* of $\hat{\mathbf{x}}$ is defined as $\|\mathbf{x} - \hat{\mathbf{x}}\|_1$. The *expected absolute error* is $\mathbb{E}\,\|\mathbf{x} - \hat{\mathbf{x}}\|_1$ where the expectation is taken over the randomness of $\hat{\mathbf{x}}$. Given workload $\mathbf{W}$, the *average error* on $\mathbf{W}$ is $\frac{1}{m}\|\mathbf{W}(\mathbf{x}) - \mathbf{W}(\hat{\mathbf{x}})\|_1$.

2.2 Private Mechanisms

Differential privacy places a bound (controlled by ϵ) on the difference in the probability of algorithm outputs for any two *neighboring* databases. For database instance I, let $nbrs(I)$ denote the set of databases differing from I in at most one record; i.e., if $I' \in nbrs(I)$, then $|(I - I') \cup (I' - I)| = 1$.

DEFINITION 2 (DIFFERENTIAL PRIVACY [9]). *A randomized algorithm $\mathcal{K}$ is ϵ-differentially private if for any instance I, any $I' \in nbrs(I)$, and any subset of outputs $S \subseteq Range(\mathcal{K})$, the following holds:*

$$Pr[\mathcal{K}(I) \in S] \leq \exp(\epsilon) \times Pr[\mathcal{K}(I') \in S]$$

Differential privacy has two important composition properties [17]. Consider k algorithms $\mathcal{K}_1, \dots, \mathcal{K}_k$, each satisfying ϵ_i-differential privacy. The *sequential* execution of $\mathcal{K}_1, \dots, \mathcal{K}_k$ satisfies $(\sum \epsilon_i)$-differential privacy. Suppose the domain is partitioned into k arbitrary disjoint subsets and $\mathcal{K}_i$ is executed on the subset of data from the i^{th} partition. The *parallel* execution of $\mathcal{K}_1, \dots, \mathcal{K}_k$ satisfies $(\max_i\{\epsilon_i\})$-differential privacy.

For functions that produce numerical outputs, differential privacy can be satisfied by adding appropriately scaled random noise to the output. The scale of the noise depends on the function's *sensitivity*, which captures the maximum difference in answers between any two neighboring databases.

DEFINITION 3 (SENSITIVITY). *Given function $f: dom(A_1) \times \dots \times dom(A_k) \rightarrow \mathbb{R}^d$, the sensitivity of f, denoted Δf, is defined as:*

$$\Delta f = \max_{I, I' \in nbrs(I)} \|f(I) - f(I')\|_1$$

Sensitivity extends naturally to a function g that operates on data vector $\mathbf{x}$ by simply considering the composition of g with the function that transforms instance I to vector $\mathbf{x}$. In this paper, we consider functions that take additional inputs from some public domain $\mathcal{R}$. For such functions, Δf measures the largest change over all pairs of neighboring databases and all $r \in \mathcal{R}$.

The Laplace mechanism achieves differential privacy by adding Laplace noise to a function's output. We use Laplace(σ) to denote the Laplace probability distribution with mean 0 and scale σ.

DEFINITION 4 (LAPLACE MECHANISM [9]). *Given function $f: dom(A_1) \times \dots \times dom(A_k) \rightarrow \mathbb{R}^d$, let $\mathbf{z}$ be a d-length vector of random variables where $z_i \sim Laplace(\Delta f/\epsilon)$. The Laplace mechanism $\mathcal{L}$ is defined as $\mathcal{L}(I) = f(I) + \mathbf{z}$.*

3. PRIVATE PARTITIONING

This section describes the first stage of the DAWA algorithm. The output of this stage is a partition B. In Sec. 3.1, we motivate the problem of finding a good partition and argue that the quality of a partition depends on the data. We then describe a differentially private algorithm for finding a good partition in Sec. 3.2.

This stage of DAWA is not tuned to the workload of queries and instead tries to select buckets such that, after statistics have been computed for the buckets and the histogram is uniformly expanded, the resulting $\hat{\mathbf{x}}$ is as close to $\mathbf{x}$ as possible.

3.1 Cost of a partition

Recall that after the partition $B = \{b_1 \dots b_k\}$ has been selected, corresponding statistics $s_1, \dots, s_k$ are computed. Let $s_i = b_i(\mathbf{x}) + Z_i$ where Z_i is a random variable representing the noise added to ensure privacy. (This noise is added in the second stage of DAWA.) Once computed, the statistics are uniformly expanded into $\hat{\mathbf{x}} = expand(B, s)$, which is an estimate for $\mathbf{x}$. If bucket b_i spans the interval $[j_1, j_2]$ we use $j \in b_i$ to denote $j \in [j_1, j_2]$. After applying uniform expansion, the resulting estimate for x_j, for $j \in b_i$, is:

$$\hat{x}_j = \frac{b_i(\mathbf{x})}{|b_i|} + \frac{Z_i}{|b_i|} \quad (1)$$

The accuracy of the estimate depends on two factors. The first factor is the bucket size. Since the scale of Z_i is fixed, larger buckets have less noise per individual $\hat{x}_j$. The second factor is the degree of uniformity within the bucket. Uniform buckets, where each x_j is near the mean of the bucket $\frac{b_i(\mathbf{x})}{|b_i|}$, yield more accurate estimates.

We can translate these observations about $\hat{x}_j$ into a bound on the expected error of $\hat{\mathbf{x}}$. For bucket b_i, let dev be a function that measures the amount the bucket *deviates* from being perfectly uniform:

$$dev(\mathbf{x}, b_i) = \sum_{j \in b_i} \left| x_j - \frac{b_i(\mathbf{x})}{|b_i|} \right| \qquad (2)$$

The bound on the expected error of $\hat{\mathbf{x}}$ is in terms of the deviation and the error due to added noise.

PROPOSITION 1. *Given histogram $H = (B, \mathbf{s})$ where $|B| = k$ and for $i = 1 \ldots k$, $s_i = b_i(\mathbf{x}) + Z_i$ where Z_i is a random variable. The uniform expansion, $\hat{\mathbf{x}} = expand(B, \mathbf{s})$, has expected error*

$$\mathbb{E}\left\|\hat{\mathbf{x}} - \mathbf{x}\right\|_1 \leq \sum_{i=1}^{k} dev(\mathbf{x}, b_i) + \sum_{i=1}^{k} \mathbb{E}|Z_i| \qquad (3)$$

The proof of this bound follows from (1) and the fact that $|a + b| \leq |a| + |b|$. Proof of a similar result is given in Acs et al. [3].

Prop. 1 reveals that the expected error of a histogram can be decomposed into two components: (a) *approximation error* due to approximating each x_j in the interval by the mean value $\frac{b_i(\mathbf{x})}{|b_i|}$ and (b) *perturbation error* due to the addition of random noise. The perturbation component is in terms of random variables Z_i, which are not fully determined until the second stage of DAWA. For the moment, let us make the simplifying assumption that the second stage uses the Laplace mechanism (with a budget of ϵ_2). Under this assumption, $Z_i \sim$ Laplace$(1/\epsilon_2)$ and $\sum_{i=1}^{k} \mathbb{E}|Z_i|$ simplifies to k/ϵ_2. This error bound conforms with our earlier intuition that we want a histogram with fewer (and therefore larger) buckets that are as uniform as possible. The optimal choice depends on the uniformity of the dataset $\mathbf{x}$ and on the budget allocated to the second stage (because smaller ϵ_2 increases perturbation error, making less uniform buckets relatively more tolerable).

We use Prop. 1 as the basis for a cost function.

DEFINITION 5 (COST OF PARTITION). *Given a partition of the domain into buckets $B = \{b_1, \ldots, b_k\}$, the cost of B is*

$$pcost(\mathbf{x}, B) = \sum_{i=1}^{k} dev(\mathbf{x}, b_i) \; + \; k/\epsilon_2 \qquad (4)$$

This cost function is based on the simplifying assumption that $Z_i \sim$ Laplace$(1/\epsilon_2)$. In fact, in the DAWA algorithm, each Z_i is a weighted combination of Laplace random variables. The weights, which are tuned to the workload, are not selected until the second stage of DAWA. However, any weight selection has the property that $\mathbb{E}|Z_i| \geq 1/\epsilon_2$. This means our choice of cost function is conservative in the sense that it favors a more fine-grained partition than would be selected with full knowledge of the noise distribution.

EXAMPLE 2. *Recall the partition $B = \{b_1, b_2, b_3, b_4\}$ in Fig. 1.*

- $b_1 = [1,2]$, $\frac{b_1(\mathbf{x})}{|b_1|} = \frac{5}{2}$, $dev(\mathbf{x}, b_1) = \frac{1}{2} + \frac{1}{2} = 1$
- $b_2 = [3,3]$, $\frac{b_2(\mathbf{x})}{|b_2|} = \frac{8}{1}$, $dev(\mathbf{x}, b_2) = 0$
- $b_3 = [4,7]$, $\frac{b_3(\mathbf{x})}{|b_3|} = \frac{3}{4}$, $dev(\mathbf{x}, b_3) = \frac{1}{4} + \frac{3}{4} + \frac{5}{4} + \frac{3}{4} = 3$
- $b_4 = [8,10]$, $\frac{b_4(\mathbf{x})}{|b_4|} = \frac{10}{3}$, $dev(\mathbf{x}, b_4) = \frac{2}{3} + \frac{4}{3} + \frac{2}{3} = 2\frac{2}{3}$

Therefore, $pcost(\mathbf{x}, B) = 6\frac{2}{3} + 4/\epsilon_2$. When $\epsilon_2 = 1.0$, $pcost(\mathbf{x}, B) = 6\frac{2}{3} + 4 = 10\frac{2}{3}$. In comparison, the cost of partitioning $\mathbf{x}$ as a single bucket $[1, 10]$ leads to a deviation of 17.2 and total pcost of 18.2. Thus B is a lower cost partition and intuitively it captures the structure of $\mathbf{x}$ which has four regions of roughly uniform density. But note that with a more stringent privacy budget of $\epsilon_2 = 0.1$, the perturbation error per bucket rises so $pcost(\mathbf{x}, B) = 6\frac{2}{3} + 40 = 46\frac{2}{3}$ whereas the pcost of a single bucket is only $17.2 + 10 = 27.2$.

Given this cost function, we can now formally state the problem that the first stage of DAWA aims to solve.

PROBLEM 1 (LEAST COST PARTITION PROBLEM). *The least cost partition problem is to find the partition that minimizes the following objective:*

$$\begin{array}{ll} \underset{B \subseteq \mathcal{B}}{\textit{minimize}} & pcost(\mathbf{x}, B) \\ \textit{subject to} & \bigcup_{b \in B} b = [1, n], \; \textit{and} \; \forall \; b, b' \in B, b \cap b' = \varnothing \end{array}$$

where $\mathcal{B}$ is the set of all possible intervals $\mathcal{B} = \{[i, j] \mid 1 \leq i \leq j \leq n\}$ and the constraint ensures that B partitions [1,n].

The next section describes our algorithm for solving this optimization problem in a differentially private manner.

3.2 Finding a least cost partition

Since partition cost is data-dependent, we cannot solve Problem 1 exactly without violating privacy. Instead, we must introduce sufficient randomness to ensure differential privacy. Our approach is efficient and simple; our main contribution is in showing that this simple approach is in fact differentially private.

Our approach is based on the observation that the cost of a partition decomposes into a cost per bucket. Let *bcost* be a function that measures the cost of an individual bucket b,

$$bcost(\mathbf{x}, b) = dev(\mathbf{x}, b) + 1/\epsilon_2.$$

For any partition B, the partition cost is simply the sum of the bucket costs: $pcost(\mathbf{x}, B) = \sum_{b \in B} bcost(\mathbf{x}, b)$. Since one needs to interact with the private database in computing the cost of each bucket, reporting the partition with the least cost will violate differential privacy. Instead, we solve Problem 1 using *noisy* cost: the noisy cost of a bucket comes from perturbing its bucket cost with a random variable sampled from the Laplace distribution, and the noisy partition cost is the sum of the noisy bucket costs.

The algorithm for this stage is shown in Algorithm 1. It takes as input the private database $\mathbf{x}$ as well as ϵ_1 and ϵ_2. The parameter ϵ_1 represents the privacy budget allocated to this stage. The parameter ϵ_2 represents the privacy budget allocated to the second stage (Algorithm 2, Sec. 4). That parameter is needed here because the value of ϵ_2 is used in calculating the bucket costs.

Algorithm 1 has three simple steps. First, it calls the subroutine ALLCOSTS to efficiently compute the cost for all possible buckets (details are below). Second, it adds noise to each bucket cost. Finally, it calls the LEASTCOSTPARTITION subroutine to find the partition with the least *noisy* cost. This is done using dynamic programming, much like classical algorithms for v-optimal histograms [14].

We analyze Algorithm 1 along three key dimensions: accuracy, computational efficiency, and privacy.

Accuracy. Accuracy is measured in terms of the difference in cost between the selected partition and the optimal choice (ignoring privacy). We give the following bound on the algorithm's accuracy.

Algorithm 1 Private partition for intervals and L_1 cost function

```
procedure PRIVATE PARTITION($\mathbf{x}$, $\epsilon_1$, $\epsilon_2$)
    // Let $\mathcal{B}$ be the set of all intervals on $[1, n]$
    // Compute cost $bcost(\mathbf{x}, b)$ for all $b \in \mathcal{B}$
    $cost \leftarrow$ ALLCOSTS($\mathbf{x}, \epsilon_2$)
    // Add noise to each bucket cost
    for $b \in \mathcal{B}$ do
        $cost[b] \leftarrow cost[b] + Z$, where $Z \sim$ Laplace$(2\Delta bcost/\epsilon_1)$
    end for
    // Find $B$ with lowest total cost based on noisy bucket costs
    // stored in $cost$
    $B \leftarrow$ LEASTCOSTPARTITION$(\mathcal{B}, cost)$
    return $B$
end procedure
```

THEOREM 1. *With probability at least $1 - \delta$, Algorithm 1 returns a solution with cost at most $OPT + t$ where OPT is the cost of the least cost solution and $t = 4\Delta c\, n \log(|\mathcal{B}|/\delta)/\epsilon_1$.*

In addition to a theoretical analysis, we do an extensive empirical evaluation in Sec. 5.

Efficiency. The computationally challenging part is ALLCOSTS, which computes the cost for each bucket. Unlike the bucket cost for a v-optimal histogram (which is based on an L_2 metric, rather than the L_1 metric used here), the cost does not decompose easily into sum and sum of square terms that can be precomputed. Nevertheless, we show that we can decompose the the cost into partial sums of x_j which can be computed using a balanced tree.

Given bucket b_i, let us identify the indexes $j \in b_i$ that are above the bucket mean, $\frac{b_i(\mathbf{x})}{|b_i|}$. Let $I^+ = \left\{ j \mid j \in b_i \text{ and } x_j \geq \frac{b_i(\mathbf{x})}{|b_i|} \right\}$. Let I^- be those below the mean, $I^- = b_i - I^+$. We can simplify $dev(\mathbf{x}, b_i)$ as follows:

$$\begin{aligned} dev(\mathbf{x}, b_i) &= \sum_{j \in I^+} \left(x_j - \frac{b_i(\mathbf{x})}{|b_i|} \right) + \sum_{j \in I^-} \left(\frac{b_i(\mathbf{x})}{|b_i|} - x_j \right) \\ &= 2 \sum_{j \in I^+} \left(x_j - \frac{b_i(\mathbf{x})}{|b_i|} \right) \\ &= 2 \sum_{j \in I^+} x_j - |I^+| \cdot \frac{b_i(\mathbf{x})}{|b_i|} \end{aligned}$$

The second equality follows from the fact that the sum of deviations above the mean must be equal to the sum of deviations below the mean. The above equation implies that the total deviation can be computed knowing only the sum of x_j for $j \in I^+$ and the size of I^+. Those quantities can be efficiently computed using a binary search tree of $x_{j_1}, \ldots, x_{j_2}$. Each node in the tree stores a value (some x_j) as well as the sum of all values in its subtree, and the number of nodes in its subtree. For any constant a, we can then compute $\sum_{j \in b_i, x_j \geq a}(x_j - a)$ via binary search.

To compute the bucket costs for all intervals with length ℓ, we can dynamically update the search tree. After the cost for interval $[j, j + \ell]$ has been computed, we can update the tree to compute interval $[j + 1, j + \ell + 1]$ by removing x_j from the tree and adding $x_{j+\ell+1}$. Using a self-balancing tree, computing all intervals of size ℓ requires $O(n \log n)$ time. To compute *all* intervals, the total runtime is $O(n^2 \log n)$.

We can reduce the runtime to $O(n \log^2 n)$ by restricting to intervals whose length is a power of two. This restriction has the potential to exclude the optimal solution. Empirically, we find that Algorithm 1 remains almost as accurate as when it uses all intervals, and is always more accurate than competing techniques (Sec. 5.3.2). The benefit of the approximation is reduced runtime, which makes it feasible to run on larger datasets.

The last step of Algorithm 1, LEASTCOSTPARTITION, is efficient, requiring time linear in n and the number of buckets.

Privacy. The proof of privacy is the main challenge. Analyzing the privacy requires some subtlety because the noise by itself is not necessarily enough to guarantee privacy. (If the Laplace mechanism was used to publish noisy costs for *all* buckets, the scale of the noise would be $\Omega(n)$.) However, when the actual noisy costs are kept secret and the only published output is the partition with the least (noisy) cost, then a small amount of noise is sufficient to ensure privacy. The noise is proportional to the sensitivity of the bucket cost. It can be shown $\Delta bcost \leq 2$.

THEOREM 2. *Algorithm 1 is ϵ_1-differentially private.*

PROOF OF THEOREM 2. Recall that $\mathcal{B} = \{[i, j] \mid 1 \leq i \leq j \leq n\}$ is the set of all intervals. For convenience, we make a few small adjustments to notation. First, we index this set: let $\mathcal{B} = \{b_1, \ldots, b_M\}$ where $M = |\mathcal{B}|$. Second, we describe a partition B in terms of this indexed set, so we say $B = \{i_1, \ldots, i_k\}$ to mean that B consists of the intervals $b_{i_1}, \ldots, b_{i_k}$. A partition B is valid if it covers the domain and its buckets are disjoint. Let $\mathcal{P}$ be the set of all valid partitions. Finally, we use this same indexing for the random variables that represent the added noise: let $\mathbf{Z} = (Z_1, \ldots, Z_M)$ where for each $i \in [1, M]$, the random variable $Z_i \sim$ Laplace(λ) represents the noise added to the cost of b_i.

Let $\mathbf{x}_0, \mathbf{x}_1$ be any pair of neighboring databases and let $B \in \mathcal{P}$ be any output of the algorithm. It suffices to prove

$$P(\mathcal{A}(\mathbf{x}_1) = B) \geq e^{-\epsilon} P(\mathcal{A}(\mathbf{x}_0) = B)$$

where $\mathcal{A}(\mathbf{x})$ denotes Algorithm 1 running on input $\mathbf{x}$ and the probability distribution is over random variables $\mathbf{Z}$.

When run on input $\mathbf{x}$, the algorithm will output partition B if and only if B is the partition with the lowest noisy cost. Formally, let $\mathbf{z} \in \mathbb{R}^M$ represent an assignment of $\mathbf{Z}$. Partition B will be selected if and only if $\mathbf{z}$ satisfies the following condition:

$$\sum_{j \in B} bcost(\mathbf{x}, b_j) + z_j < \min_{B' \in \mathcal{P} - \{B\}} \left\{ \sum_{k \in B'} bcost(\mathbf{x}, b_k) + z_k \right\}$$

Since neighboring databases $\mathbf{x}_0$ and $\mathbf{x}_1$ only differ by one record and the buckets of B partition the domain, there must be exactly one $i \in B$ where $bcost(\mathbf{x}_0, b_i) \neq bcost(\mathbf{x}_1, b_i)$. We will now derive an expression for the probability that B is selected that focuses on the noisy cost of bucket b_i. To do this, it will be convenient to partition the space of possible partitions into those that include bucket b_i and those that do not. Let $\mathcal{P}^+ = \{B \mid B \in \mathcal{P} \text{ and } i \in B\}$ and let $\mathcal{P}^- = \mathcal{P} - \mathcal{P}^+$. B will be selected if and only if (a) B is the partition with least noisy cost in $\mathcal{P}^+$ and (b) B has lower noisy cost than any partition in $\mathcal{P}^-$. We examine these two conditions in turn.

For condition (a), observe that all partitions in $\mathcal{P}^+$ use bucket b_i, thus whether (a) holds is independent of the outcome of Z_i since it has the same effect on the scores of all partitions in $\mathcal{P}^+$. We use $\mathbf{z}^{-i}$ as shorthand for $(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)$. Let ϕ be a predicate that is true if and only if the assignment of $\mathbf{z}^{-i}$ makes B the least cost partition among $\mathcal{P}^+$, and false otherwise:

$$\phi(\mathbf{x}, \mathbf{z}^{-i})$$

$$= \sum_{j \in B - \{i\}} bcost(\mathbf{x}, b_j) + z_j < \min_{B' \in \mathcal{P}^+ - \{B\}} \left\{ \sum_{k \in B' - \{i\}} bcost(\mathbf{x}, b_k) + z_k \right\}$$

Since $\mathbf{x}_0$ and $\mathbf{x}_1$ only differ in the score assigned to bucket b_i, $\phi(\mathbf{x}_0, \mathbf{z}^{-i}) = \phi(\mathbf{x}_1, \mathbf{z}^{-i})$ for all $\mathbf{z}^{-i} \in \mathbb{R}^{M-1}$.

For condition (b), let ψ be a predicate that is true if and only if the assignment of $\mathbf{z}$ makes B a lower cost partition than any partition in $\mathcal{P}^-$, and false otherwise. A key insight is that if we fix $\mathbf{z}^{-i}$, then B will have lower cost provided that z_i is small enough.

$$\psi(\mathbf{x},\mathbf{z}) = \sum_{j\in B} bcost(\mathbf{x},b_j) + z_j < \min_{B'\in\mathcal{P}^-}\left\{\sum_{k\in B'} bcost(\mathbf{x},b_k) + z_k\right\}$$
$$= z_i < Z(\mathbf{x},\mathbf{z}^{-i})$$

where

$$Z(\mathbf{x},\mathbf{z}^{-i}) =$$
$$\min_{B'\in\mathcal{P}^-}\left\{\sum_{k\in B'} bcost(\mathbf{x},b_k) + z_k\right\} - \sum_{j\in B} bcost(\mathbf{x},b_j) - \sum_{\ell\in B-\{i\}} z_\ell$$

The upper bound $Z(\mathbf{x},\mathbf{z}^{-i})$ depends on the database. For neighboring databases $\mathbf{x}_0$ and $\mathbf{x}_1$, $Z(\mathbf{x}_1,\mathbf{z}^{-i}) \geq Z(\mathbf{x}_0,\mathbf{z}^{-i}) - 2\Delta bcost$. This is because compared to the score on $\mathbf{x}_0$, the score on neighboring database $\mathbf{x}_1$ of the minimum cost partition in $\mathcal{P}^-$ could be lower by at most $\Delta bcost$ and the cost of B could be larger by at most $\Delta bcost$.

We can now express the probability that the algorithm on input $\mathbf{x}$ outputs B in terms of ϕ and ψ. Let $f_{\mathbf{Z}}$ (respectively f_Z) denote the density function for a multivariate (respectively univariate) Laplace random variable, and $\mathbf{I}[\cdot]$ denote the indicator function.

$$P(\mathcal{A}(\mathbf{x}) = B) = P(\phi(\mathbf{x},\mathbf{Z}^{-i}) \wedge \psi(\mathbf{x},\mathbf{Z}))$$
$$= \int \mathbf{I}\left[\phi(\mathbf{x},\mathbf{z}^{-i}) \wedge \psi(\mathbf{x},\mathbf{z})\right] f_{\mathbf{Z}}(\mathbf{z})\mathrm{d}\mathbf{z}$$
$$= \int \mathbf{I}\left[\phi(\mathbf{x},\mathbf{z}^{-i})\right] f_{\mathbf{Z}^{-i}}(\mathbf{z}^{-i}) \left(\int \mathbf{I}\left[\psi(\mathbf{x},\mathbf{z})\right] f_{Z_i}(z_i)\mathrm{d}z_i\right) \mathrm{d}\mathbf{z}^{-i}$$
$$= \int \mathbf{I}\left[\phi(\mathbf{x},\mathbf{z}^{-i})\right] f_{\mathbf{Z}^{-i}}(\mathbf{z}^{-i}) P(Z_i < Z(\mathbf{x},\mathbf{z}^{-i}))\mathrm{d}\mathbf{z}^{-i}$$

Since $P(Z_i < C)$ decreases with decreasing C, we have for neighboring databases $\mathbf{x}_0$ and $\mathbf{x}_1$ and any $\mathbf{z}^{-i} \in \mathbb{R}^{M-1}$ that

$$P(Z_i < Z(\mathbf{x}_1,\mathbf{z}^{-i}))$$
$$\geq P(Z_i < Z(\mathbf{x}_0,\mathbf{z}^{-i}) - 2\Delta bcost)$$
$$\geq e^{-2\Delta bcost/\lambda} P(Z_i < Z(\mathbf{x}_0,\mathbf{z}^{-i}))$$

where the last line follows from the fact that if Z is a Laplace random variable with scale λ, then for any z and any constant $c > 0$, $P(Z < z - c) \geq e^{-c/\lambda} P(Z < z)$.

In addition, we observed earlier that $\phi(\mathbf{x}_0,\mathbf{z}^{-i}) = \phi(\mathbf{x}_1,\mathbf{z}^{-i})$ for all $\mathbf{z}^{-i} \in \mathbb{R}^{M-1}$. Therefore, we can express a lower bound for $P(\mathcal{A}(\mathbf{x}_1) = H)$ strictly in terms of $\mathbf{x}_0$:

$$P(\mathcal{A}(\mathbf{x}_1) = B)$$
$$\geq \int \mathbf{I}\left[\phi(\mathbf{x}_0,\mathbf{z}^{-i})\right] f_{\mathbf{Z}^{-i}}(\mathbf{z}^{-i}) e^{-2\Delta bcost/\lambda} P(Z_i < Z(\mathbf{x}_0,\mathbf{z}^{-i}))\mathrm{d}\mathbf{z}^{-i}$$
$$= e^{-2\Delta bcost/\lambda} P(\mathcal{A}(\mathbf{x}_0) = B) = e^{-\epsilon} P(\mathcal{A}(\mathbf{x}_0) = B)$$

since, according the algorithm description, $\lambda = 2\Delta bcost/\epsilon$. □

Remark In Algorithm 1 we can reduce the noise from $2\Delta bcost$ to $\Delta bcost$ plus the sensitivity of the particular bucket. The benefit is a reduction in noise (by at most a factor of 2) for some buckets. This optimization is used in the experiments.

4. PRIVATE BUCKET COUNT ESTIMATION

This section describes the second stage of the DAWA algorithm. Given the partition $B = \{b_1, \ldots, b_k\}$ determined by the first stage of DAWA, it remains to privately estimate counts for each bucket, using budget ϵ_2. Thus the goal of the second stage is to produce $\mathbf{s} = s_1 \ldots s_k$. Naive solutions like adding Laplace noise to each bucket count result in high error for many workloads. In this section, we show how to adapt the existing framework of the matrix mechanism [15] to create a workload-adaptive algorithm for computing the bucket counts. Within this framework, we describe a novel greedy algorithm for minimizing error of the workload queries.

4.1 Workload-adaptive bucket estimation

Our approach relies on the matrix mechanism [15], which provides a framework for answering a batch of linear queries (i.e. a workload). Instead of answering the workload directly, the matrix mechanism poses another set of queries, called the query strategy, and uses the Laplace mechanism to obtain noisy answers. These noisy answers can then be used to derive an estimated data vector using ordinary least squares. The answers to the workload can then be computed from the estimated data vector.

Adapting the matrix mechanism to the private estimation of the bucket counts entails two challenges. First, our original workload $\mathbf{W}$ is expressed in terms of $\mathbf{x}$, but we seek a mechanism that produces estimates of the bucket counts $\mathbf{s}$. Below, in Sec. 4.2, we describe a transformation of $\mathbf{W}$ into a new workload in terms of the domain of buckets that allows us to optimize error for the original $\mathbf{W}$. Second is the familiar challenge of the matrix mechanism: computing a query strategy, suited to the given workload but not dependent on the data, so as to minimize the mean square error of answering the workload. In general, computing the query strategy that minimizes error under the matrix mechanism requires solving high complexity optimization problems [15,24]. Hence we extend ideas from prior work [16, 23] that efficiently compute approximately optimal query strategies. We fix a template strategy that is well-suited for anticipated workloads, but then compute approximately optimal weighting factors to emphasize the strategy queries that matter most to the workload. This effectively adjusts the privacy budget to maximize accuracy on the given workload. Since our anticipated workload consists of range queries, we use a hierarchical query strategy as a template, similar to prior work [6,13,20].

Our goal is to minimize the mean squared error of answering the workload by assigning different scaling to queries. Although similar goals are considered in prior works, their methods impose additional constraints that do not apply in our setting: Li and Miklau [16] require the number of strategy queries to be no more than the domain size; and Yaroslavtsev et al. [23] require a fixed "recovery" matrix to derive workload answers from strategy answers.

4.2 Workload transformation

Recall that, given statistics $\mathbf{s} = s_1 \ldots s_k$ for the buckets in B, we answer the workload $\mathbf{W}$ by first uniformly expanding $\mathbf{s}$ into an estimate $\hat{\mathbf{x}}$ and then computing $\mathbf{W}(\hat{\mathbf{x}})$. We now show an equivalent formulation for this process by transforming the $m \times n$ workload $\mathbf{W}$ into a new $m \times k$ workload $\hat{W}$ such that the workload query answers can be computed directly as $\hat{W}(\mathbf{s})$. The most important consequence of this transformation is that we need not consider the uniform expansion step while adapting the estimation of $\mathbf{s}$ to the workload. Since $k < n$, an additional convenient consequence is that the domain size is typically reduced so that the complexity of the matrix mechanism operations is lower[1].

DEFINITION 6 (QUERY TRANSFORMATION). *Given a query $q = (q_1, \ldots, q_n)$, defined on data vector $\mathbf{x}$, and given partition*

[1] Table 1 in Sec. 5 shows the domain sizes of the derived partitions for each example dataset we consider in the performance analysis.

B, the transformation of q with respect to B is defined as $\hat{q} = (\hat{q}_1, \ldots, \hat{q}_k)$ where

$$\hat{q}_j = \frac{1}{|b_j|} \sum_{i \in b_j} q_i.$$

EXAMPLE 3. *Recall the partition $B = \{b_1, b_2, b_3, b_4\}$ in Fig. 1. Consider range query $q = x_2 + x_3 + x_4 + x_5 + x_6$. This query can be reformulated in terms of the statistics of the three buckets b_1 to b_3 that cover the range spanned by q. Hence $\hat{q} = \frac{1}{2}s_1 + s_2 + \frac{3}{4}s_3$.*

Accordingly, a transformed workload, $\hat{W}$, is formed by transforming each query in $\mathbf{W}$.

PROPOSITION 2. *For any workload $\mathbf{W}$ and buckets B, let $\hat{W}$ be the transformation of $\mathbf{W}$. Then for any statistics $\mathbf{s}$ for B,*

$$\mathbf{W}(expand(B, \mathbf{s})) = \hat{W}(\mathbf{s})$$

We now seek a private estimation procedure for the bucket counts in $\mathbf{s}$ that is adapted to the transformed workload $\hat{W}$. It follows from Prop. 2 that minimizing the error for workload $\hat{W}$ will also result in minimum error for $\mathbf{W}$ after expansion.

4.3 Optimal scaling of hierarchical queries

As mentioned above, our template strategy, denoted as Y, is a set of interval queries that forms a tree with branching factor t. The queries at the leaves are individual entries of $\mathbf{s}$. For each higher level of the tree, interval queries of t nodes in the previous level are aggregated, and the aggregated query becomes their parent node in the upper level. Since the number of nodes at each level may not be a multiple of t, the last nodes at each level are allowed to have fewer than t children. This aggregating process is repeated until the topmost level only has one node, whose interval is the entire domain $\mathbf{s}$.

For each query $q \in Y$, let c_q be the scaling q, and Y_c be the set of queries in Y after the scaling. The goal of this stage of our algorithm is to find a scaling of Y_c to minimize the total squared error of answering $\hat{W}$ using Y_c. According to [15], scaling up all queries in Y_c by a positive constant does not change the mean squared error of answering any query using Y_c. Thus, without loss of generality, we bound the sensitivity of Y_c by 1 by requiring that,

$$\sum_{q(i) \neq 0, q \in Y} c_q \leq 1, \quad i = 1, \ldots, k. \tag{5}$$

When the sensitivity of Y_c is fixed, the scaling controls the accuracy with which the query will be answered: the larger scaling leads to the more accurate answer. Let the matrix representation of Y be $\mathbf{Y}$ and $\mathbf{D}_Y$ be the diagonal matrix whose diagonal entries are scales of queries in Y_c. Then the matrix form of Y_c can be represented as $\mathbf{D}_Y\mathbf{Y}$. Since the sensitivity of Y_c is bounded by 1, according to [15], the squared error of answering a query q (with matrix form $\mathbf{q}$) using Y_c under ϵ_2 differential privacy is:

$$\frac{2}{\epsilon_2^2}\mathbf{q}^T((\mathbf{D}_Y\mathbf{Y})^T\mathbf{D}_Y\mathbf{Y})^{-1}\mathbf{q} = \frac{2}{\epsilon_2^2}\mathbf{q}^T(\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y})^{-1}\mathbf{q}.$$

Let $\hat{\mathbf{W}}$ be the matrix form of $\hat{W}$. The total squared error of answering all queries in $\mathbf{W}$ can then be computed as:

$$\sum_{q \in \hat{W}} \frac{2}{\epsilon_2^2}\mathbf{q}^T(\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y})^{-1}\mathbf{q} = \frac{2}{\epsilon_2^2}tr(\hat{\mathbf{W}}(\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y})^{-1}\hat{\mathbf{W}}^T)$$
$$= \frac{2}{\epsilon_2^2}tr(\hat{\mathbf{W}}^T\hat{\mathbf{W}}(\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y})^{-1}). \tag{6}$$

Above, $tr()$ is the trace of a square matrix: the sum of all diagonal entries. We now formally state the query scaling problem.

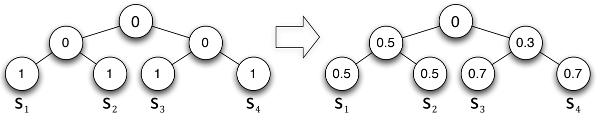


Figure 2: Scaling allocation for the second level. $\lambda = 0.5$ at the left node, and $\lambda = 0.3$ at the right node.

PROBLEM 2 (OPTIMAL QUERY SCALING PROBLEM). *The optimal query scaling problem is to find a c_q for each query $q \in Y$ that minimizes Eqn (6) under the constraint of Eqn (5).*

4.4 Efficient greedy scaling

Problem 2 is a significant simplification of the general strategy selection problem from [15] because the template strategy is fixed and only scaling factors need to be computed. Nevertheless the optimal solution to Problem 2 appears difficult since equation (6) is still non-convex. Instead of pursuing an optimal solution, we solve the problem approximately using the following greedy algorithm. The algorithm works in a bottom-up manner. It initially puts scale 1 to all leaf queries in Y and 0 to all other queries in Y. For query $q \in Y$, the algorithm chooses a $\lambda_q \in [0, 1]$. The scaling is then reallocated as follows: the scaling on each of its descendent q' is reduced from $c_{q'}$ to $(1 - \lambda_q)c_{q'}$ and the scaling on q is λ_q. The value of λ_q is chosen to minimize Equation (6) after the scaling reallocation. Notice that the new scaling still satisfies the constraint in equation (5).

EXAMPLE 4. *An example of the scaling reallocation is shown in Fig. 2, in which two different λ are chosen for two nodes (queries) at the second level.*

When the scaling reallocation terminates, the algorithm asks all scaled queries and adds Laplace$(1/\epsilon_2)$ noise to the answer of each query. After that, any inconsistencies among those noisy query answers are resolved using ordinary least squares inference.

The major challenge in the algorithm described above is to efficiently choose λ_q in each step. To simplify the presentation, we always assume the branching factor $t = 2$, though the discussion is valid for any branching factor.

For each interval query $q \in Y$, let $[i, j]$ be the corresponding interval of q. Use $\hat{\mathbf{W}}_q$ to denote the matrix consisting of the i^{th} to j^{th} column of $\hat{\mathbf{W}}$, and $\mathbf{Y}_q$ to denote the matrix consisting of the i^{th} to j^{th} column of the matrix of queries in the subtree of rooted at q. Let $\mathbf{D}_q$ be the diagonal matrix whose diagonal entries are c'_q for all q' in the subtree rooted at q. For each query $q \in Y$ that is not on a leaf of Y, let q_1, q_2 be queries of its child nodes.

For each query $q \in Y$ that is not on a leaf of Y, according to the construction of Y, $q = q_1 + q_2$. Hence $\hat{\mathbf{W}}_q = [\hat{\mathbf{W}}_{q_1} \; \hat{\mathbf{W}}_{q_2}]$. Futher, since the queries in the subtree of q are the union of queries in the subtree of q_1, q_2, as well as query q itself, for a given λ_q,

$$\mathbf{D}_q = \begin{bmatrix} \lambda_q & 0 & 0 \\ 0 & (1-\lambda_q)\mathbf{D}_{q_1} & 0 \\ 0 & 0 & (1-\lambda_q)\mathbf{D}_{q_2} \end{bmatrix}.$$

When choosing a λ_q, due to the fact that the scalings on all ancestors of q in Y are 0 at this moment, the matrix $\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y}$ becomes a block diagonal matrix, and $\mathbf{Y}_q^T\mathbf{D}_q^2\mathbf{Y}_q$ is one of its blocks. Therefore, the choice of λ_q only depends on $\hat{\mathbf{W}}_q$ and Y_q, which means λ_q can be determined locally, by minimizing

$$tr(\hat{\mathbf{W}}_q^T\hat{\mathbf{W}}_q(\mathbf{Y}_q^T\mathbf{D}_q^2\mathbf{Y}_q)^{-1}). \tag{7}$$

Since the only unknown variable in Eqn. (7) is λ_q, solving its optimal solution is much easier than solving the optimal scaling for

Algorithm 2 Estimating bucket counts $\mathbf{s}$.

```
procedure BucketCountEstimator($B$, $W$, $\mathbf{x}$, $\epsilon_2$)
    Given workload $W$ and buckets $B$, transform workload to $\hat{W}$
    Let $Y$ be a tree of queries over buckets
    For each query $q \in Y$, let $c_q = 1$ if $q$ is a leaf, and $c_q = 0$ otherwise.
    for all $q \in Y$, from bottom to top do
        Numerically find $\lambda_q$ that minimizing Equation (9).
        Let $c_q = \lambda_q$.
        For each descendent $q'$ of $q$, let $c_{q'} = (1 - \lambda_q)c_{q'}$.
    end for
    Let $\mathbf{y}$ be the vector of $c_q q(B(\mathbf{x})) + \text{Laplace}(1/\epsilon_2)$ for all $q \in Y$.
    return $\mathbf{s} = (\mathbf{Y}^T\mathbf{D}_Y^2\mathbf{Y})^{-1}(\mathbf{D}_Y\mathbf{Y})^T\mathbf{y}$
end procedure
```

all queries in Eqn. (6). However, one of the problems of choosing λ_q using equation (7) is that it is biased towards q and λ_q is larger than required. When deciding the scaling on a query $q \in Y$, the scalings on all the ancestors of q are 0. Hence the scaling distribution is based on the assumption that all queries that contain q are answered by q, which is not true after some ancestors of q are assigned non-zero scalings.

In order to reduce this bias, a heuristic decay factor μ is introduced to control the impact of q on queries that need to be answered with q. The following matrix is used in equation (7) to take the place of $\hat{\mathbf{W}}_q^T\hat{\mathbf{W}}_q$:

$$\mu\hat{\mathbf{W}}_q^T\hat{\mathbf{W}}_q + (1-\mu)\begin{bmatrix} \hat{\mathbf{W}}_{q_1}^T\hat{\mathbf{W}}_{q_1} & 0 \\ 0 & \hat{\mathbf{W}}_{q_2}^T\hat{\mathbf{W}}_{q_2} \end{bmatrix}. \qquad (8)$$

As above, the bias of equation (7) comes from the assumption that the scalings on all the ancestors of q are 0. Hence there will be less bias when q is more close to the root of Y. In our implementation, μ is set to be $t^{-\frac{l}{2}}$ where t is the branching factor of Y and l is the depth of q in Y. Our algorithm then minimizes the following quantity instead of equation (7).

$$tr\left(\left(t^{-\frac{l}{2}}\hat{\mathbf{W}}_q^T\hat{\mathbf{W}}_q+(1-t^{-\frac{l}{2}})\begin{bmatrix} \hat{\mathbf{W}}_{q_1}^T\hat{\mathbf{W}}_{q_1} & 0 \\ 0 & \hat{\mathbf{W}}_{q_2}^T\hat{\mathbf{W}}_{q_2} \end{bmatrix}\right)(\mathbf{Y}_q^T\mathbf{D}_q^2\mathbf{Y}_q)^{-1}\right). \qquad (9)$$

At first glance, computing equation (9) seems complicated since $(\mathbf{Y}_q^T\mathbf{D}_q^2\mathbf{Y}_q)^{-1}$ needs to be recomputed for each λ_q. However, if we record some quantities in the previous step, it only takes $O(m(j-i+1)+(j-i+1)^2)$ time for the preprocessing and Eqn. (9) can be computed in $O(1)$ time for any λ_q. We omit the details due to the lack of space.

The entire process of computing bucket statistics is summarized in Algorithm 2. Notice that Algorithm 2 just chooses a scaling Y_c with sensitivity at most 1, and answers $\mathbf{s}$ using Y_c as a strategy. The privacy guarantee of Algorithm 2 follows from that of the matrix mechanism.

Proposition 3. *Algorithm 2 is ϵ_2-differentially private.*

Theorem 3. *Algorithm 2 takes $O(mk\log k + k^2)$ time. In the worst case, $k = O(n)$, and Algorithm 2 takes $O(mn\log n + n^2)$ time.*

Hence, Algorithm 2 costs much less time than previous general query selection approaches in the matrix mechanism [15, 24].

5. EXPERIMENTAL EVALUATION

We now evaluate the performance of DAWA on multiple datasets and workloads, comparing it with recently-proposed algorithms (in Sec. 5.2). We also examine the effectiveness of each of the two main steps of our algorithm (Sec. 5.3). Finally, we consider an extension of our technique to two-dimensional spatial data and compare it with state-of-the-art algorithms (Sec. 5.4).

5.1 Experimental setup

In the experiments that follow, the primary metric for evaluation is the average L_1 error per query for answering the given workload queries. Most workloads we use are generated randomly (as described below). Each experimental configuration is repeated on 5 random workloads with 3 trials for each workload. The results reported are the average across workloads and trials. The random workloads are generated once and used for all experiments.

The privacy budget in DAWA is set as $\epsilon_1 = 0.25\epsilon$ and $\epsilon_2 = 0.75\epsilon$. Unless otherwise specified, the first step of DAWA constructs a partition using intervals whose lengths must be a power of 2, an approximation that is described in Sec. 3. For the second step of the algorithm, the branching factor of the query tree is set to 2.

Datasets. There are seven different 1-dimensional datasets considered in our experiments. Although these datasets are publicly available, many of them describe a type of data that could be potentially sensitive, including financial, medical, social, and search data. Adult is derived from U.S. Census data [4]: the histogram is built on the "capital loss" attribute, which is the same attribute used in [12]. Income is based on the IPUMS American community survey data from 2001-2011; the histogram attribute is personal income [18]. Medical Cost is a histogram of personal medical expenses based on a national home and hospice care survey from 2007 [19]. Nettrace is derived from an IP-level network trace collected at the gateway router of a major university. The histogram attribute is the IP address of internal hosts and so the histogram reports the reports the number of external connections made by each internal host [13]. Search Logs is a dataset extracted from search query logs that reports the frequency of the search term "Obama" over time (from 2004 to 2010) [13]. Furthermore, we consider two temporal datasets derived from two different kinds of network data. HepPh is a citation network among high energy physics pre-prints on arXiv and Patent is a citation network among a subset of US patents [1]. These last datasets describe public data but serve as a proxy for social network data, which can be highly sensitive. For both datasets, the histogram reports the number of new incoming links at each time stamp. To eliminate the impact of domain size in comparing the "hardness" of different datasets, all datasets above are aggregated so that the domain size n is 4096.

Query workloads. We run experiments on four different kinds of workloads. The *identity* workload consists of all unit-length intervals $[1,1], [2,2], \ldots, [n,n]$. The *uniform interval* workload samples 2000 interval queries uniformly at random. In addition, workloads that are not uniformly distributed over the domain are also included. The *clustered interval* workload first samples five numbers uniformly from $[1,n]$ to represent five cluster centers and then samples 400 interval queries for each cluster. Given cluster center c, an interval query is sampled as $[c-|X_\ell|, c+|X_r|]$ where X_ℓ and X_r are independent random variables from a normal distribution with a standard deviation of 256. The *large clustered interval* workload is generated in the same way but the standard deviation is 1024.

Competing algorithms. We compare DAWA with six algorithms. For data-independent algorithms, we include a simple approach (Identity) that adds Laplace noise to each entry of $\mathbf{x}$ and the Privelet algorithm [20], which is designed to answer range queries on large domains. For data-dependent algorithms, we compare with EFPA [3], P-HP [3], StructureFirst [22],[2] and MWEM [12], all of which are described in Sec. 6. For MWEM, we set the number of

[2] The other algorithms from Xu et al. [22] take more than 20 hours to complete a single trial. Therefore, they are not included.

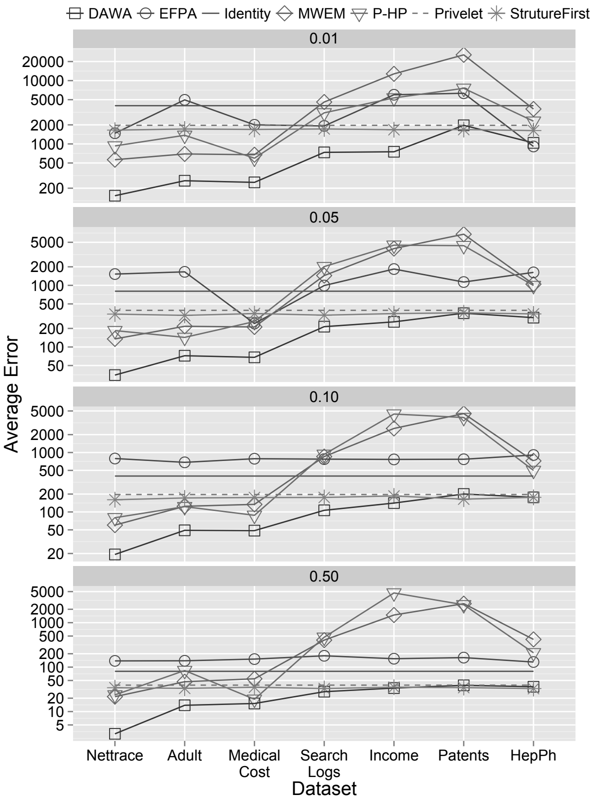


Figure 3: Average error on the *uniform intervals* workload across multiple datasets. The privacy budget ranges from $\epsilon = 0.01$ (top) to $\epsilon = 0.5$ (bottom).

Nettrace	Adult	Med. Cost	S. Logs	Income	Patents	HepPh
22	29	20	500	1537	1870	2168

Table 1: The number of buckets, k, in the optimal partition when $\epsilon = 0.1$. The original domain size is $n = 4096$ for each dataset.

iterations, T, to the value in $\{10, 20, \ldots, 190, 200\}$ that achieves the lowest error on each dataset for the *uniform intervals* workload and $\epsilon = 0.1$. We use that T for all experiments on that dataset.

With the exception of MWEM, all algorithms are quite efficient, usually finishing in seconds. MWEM slows down for harder datasets which require a high T, taking up to ten seconds on these datasets.

5.2 Accuracy for interval workloads

Fig. 3 presents the main error comparison of all algorithms on workloads of *uniform intervals* across a range of datasets and settings of ϵ. While data-independent algorithms like Privelet and Identity offer constant error across datasets, the error of data-dependent algorithms can vary significantly.[3] For some datasets, data-dependent algorithms can be much more accurate. For example, on Nettrace with $\epsilon = 0.01$, *all* of the data-dependent algorithms have lower error than the best data-independent algorithm (Privelet). For this dataset, the error of DAWA is at least an order of magnitude lower than Privelet. These results suggest the potential power of data-dependence.

(a) Smallest ratio across datasets

ϵ	Identity	Privelet	MWEM	EFPA	P-HP	S. First
0.01	2.04	1.00	2.65	0.88	2.20	0.86
0.05	2.27	1.11	3.00	3.20	1.98	1.01
0.1	2.00	0.98	2.54	3.84	1.81	0.82
0.5	2.06	1.01	3.39	3.60	1.25	0.89

(b) Largest ratio across datasets

ϵ	Identity	Privelet	MWEM	EFPA	P-HP	S.First
0.01	26.42	12.93	17.00	18.94	7.09	10.85
0.05	22.97	11.24	19.14	43.58	17.57	9.77
0.1	20.85	10.20	22.54	41.09	31.41	8.32
0.5	25.47	12.46	68.75	43.69	138.14	10.89

Table 2: Ratio of algorithm error to DAWA error, for each competing algorithm and ϵ setting on *uniform intervals*: (a) smallest ratio observed across datasets; (b) largest ratio across datasets.

There are other datasets, however, where the competing data-dependent algorithms appear to break down. In the figure, the datasets are ordered by the cost of an optimal partition (i.e., an optimal solution to Step 1 of our algorithm) when $\epsilon_2 = 0.1$. This order appears to correlate with "hardness." Datasets on the left have low partition cost and appear to be relatively "easy," presumably because data-dependent algorithms are able to exploit uniformities in the data. However, as one moves to the right, the optimal partition cost increases and the datasets appear to get more difficult. It is on many of the "harder" datasets where competing data-dependent algorithms suffer: their error is higher than even a simple baseline approach like Identity.

In contrast, DAWA does not break down when the dataset is no longer "easy." On the moderately difficult dataset Search Logs, DAWA is the only data-dependent algorithm that outperforms data-independent algorithms. On the "hardest" datasets, its performance is comparable to data independent techniques like Privelet. DAWA comes close to achieving the best of both worlds: it offers very significant improvement on easier datasets, but on hard datasets roughly matches the performance of data-independent techniques.

For the same workload, datasets, and algorithms, Table 2 reports the performance of DAWA relative to other algorithms. Each cell in the table reports the ratio of algorithm error to DAWA error. Table 2(a) reports the smallest ratio achieved over all datasets—i.e., how close the competing algorithm comes to matching, or in some cases beating, DAWA . Table 2(b) reports the largest ratio achieved—i.e., how much worse the competing algorithm can be on some dataset. Table 2(b) reveals that every competing algorithm has at least 7.09 times higher error than DAWA on some dataset.

Table 2(a) reveals that DAWA is sometimes less accurate than another algorithm, but only moderately so. This occurs on the "hardest" datasets, Patents and HepPh, where DAWA has error that is at most $\frac{1}{0.82} \approx 22\%$ higher than other approaches. On these hard datasets, the optimal partition has thousands of buckets (see Table 1), indicating that it is highly non-uniform. On non-uniform data, the first stage of the DAWA algorithm spends ϵ_1 of the privacy budget just to select a partition that is similar to the base buckets. Despite the fact that the first stage of the algorithm does not help much on "hard" datasets, DAWA is still able to perform comparably to the best data-independent technique, in contrast to the other data dependent strategies which perform poorly on such "hard" datasets.

In addition to *uniform interval* workload, we also ran experiments on the other three types of workloads. The performance of

[3] StructureFirst is an exception to this trend: its observed performance is almost totally independent of the dataset. Its partition selection algorithm uses a high sensitivity scoring function (which is based on L_2 rather than L_1). Thus, partition selection is very noisy and close to random for all datasets.

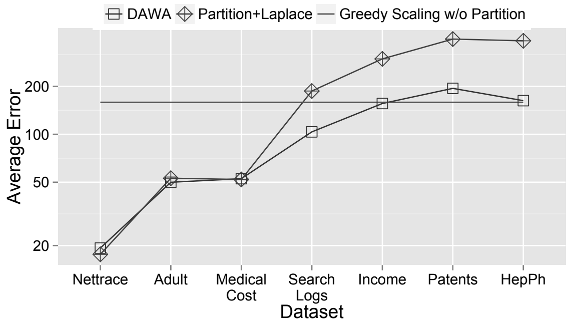


Figure 4: Average error of isolated parts of DAWA , $\epsilon = 0.1$.

DAWA relative to its competitors is qualitatively similar to the performance on *uniform interval* workload shown above. Due to limited space, the figures are omitted.

5.3 Analyzing the performance of DAWA

To further understand the strong performance shown above, we study the two steps of DAWAin detail, first by isolating the impact of each step, and then by assessing the effectiveness of the approximations made in each step.

5.3.1 Isolating the two steps

To isolate the performance of each of the two stages of the DAWA algorithm, we consider two DAWA variants. The first variant combines the first stage of DAWA with the Laplace mechanism (Partition+Laplace). This algorithm is data dependent but not workload aware. This variant has two stages like DAWA, and the budget allocation is the same: a quarter of the budget is spent on partitioning and the rest on estimating bucket counts. The second variant omits the first stage of DAWA and runs the second stage of the algorithm on the original domain (Greedy Scaling w/o Partition). For this variant, the entire budget is allocated to estimating counts. This algorithm is workload-aware but data-independent, thus its performance is the same across all datasets.

Fig. 4 shows the results. On the "easier" datasets, DAWA has much lower error than Greedy Scaling w/o Partition. For these datasets, which have large uniform regions, allocating a portion of the privacy budget to selecting a data-dependent partition can lead to significant reductions in error. In these cases, the benefit outweighs the cost. On "hard" datasets, where most data-dependent algorithms fail, the partitioning does not appear to help much. One reason may be that on these datasets even the optimal partition has many buckets (Table 1), so the partitioned dataset is not radically different from the original domain. However, even on these hard datasets, DAWA is almost as accurate as Greedy Scaling w/o Partition, suggesting that there is still enough improvement from partitioning to justify its cost.

Finally, we can examine the effect of the second stage of DAWA by comparing DAWA against Partition+Laplace. On "easy" datasets, they perform about the same. On these datasets, the partition selected in the first stage has a small number of buckets, which means that the input to the second step is a small domain. Since the Laplace mechanism works well on small domains, the lack of a performance difference is not surprising. However, on "harder" datasets, the partitions produced by the first stage have a large number of buckets and Partition+Laplace performs poorly. In such cases, using the second step of DAWA proves highly beneficial and DAWA has much lower error than Partition+Laplace.

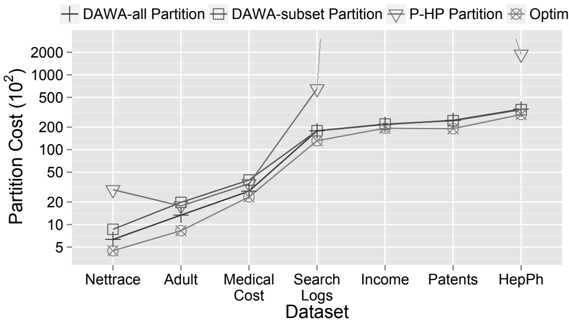


Figure 5: A comparison of alternative algorithms for the first step of DAWA with $\epsilon = 0.1$.

5.3.2 Effectiveness of partition selection

Here we evaluate the effectiveness of the first step of the DAWA algorithm, partition selection (Algorithm 1). Recall from Sec. 3.2 that it is possible to restrict the set of intervals considered in selecting the partition. We compare two versions of the algorithm: DAWA-subset only considers intervals whose lengths are a power of two, DAWA-all considers all possible intervals. We compare these variants with the optimal solution, which is computed by solving Problem 1 using the bucket cost without noise, ignoring privacy considerations. Finally, we compare with P-HP [3], which is also designed to solve Problem 1. To facilitate a fair comparison, for this experiment each algorithm spends the same amount of privacy budget on selecting the partition.

The results are shown in Fig. 5 for $\epsilon = 0.1$ where the y-axis measures the partition cost (Def. 5). We further assume $\epsilon_2 = 0.1$ when computing the cost of each bucket in DAWA . The partition cost of DAWA-all is close to optimal. The cost of the partition of DAWA-subset is sometimes higher than that of DAWA-all especially on "easier" datasets. Generally, however, DAWA-subset and DAWA-all perform similarly. This suggests that the efficiency benefit of DAWA-subset does not come at the expense of utility. The cost of the partition selected by P-HP is almost as low as the cost of the DAWA-subset partition on the Adult and Medical Cost datasets, but it is orders of magnitude larger on other datasets (on Income and Patents it is at least 1.6×10^6). This provides empirical evidence that Algorithm 1 is much more accurate than the recursive bisection approach of P-HP. The results with $\epsilon \in \{0.01, 0.05, 0.5\}$ are similar and omitted.

5.3.3 Effectiveness of adapting to workload

The second stage of DAWA designs a query strategy that is tuned to the workload, as described by Algorithm 2. Here we combine our partitioning algorithm in the first step with some alternative strategies and compare them with DAWA to evaluate the effectiveness of our greedy algorithm. Two alternative ways to scale queries in Y are considered: all queries are given the same scaling (Partition+Uniform) based on Hay et al. [13], and the scaling decreases geometrically from leaves to root (Partition+Geometric) based on Cormode et al. [6]. The Laplace mechanism (Partition+Laplace) is also included. Among the alternative algorithms, Partition+Geometric is designed to answer *uniform interval* workloads, and the Laplace mechanism is known to be the optimal data-independent mechanism for the *identity* workload. We do not consider any data-dependent techniques as alternatives for the second step. After partitioning, uniform regions in the data have been largely removed and our results show that the data-dependent algorithms perform poorly if used in this step.

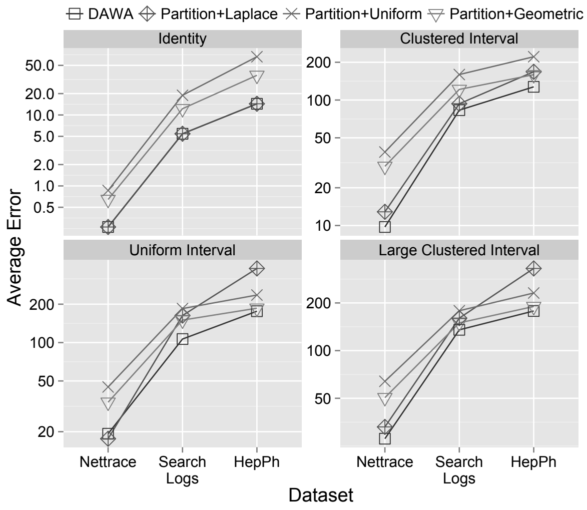


Figure 6: A comparison of alternative algorithms for the second step of DAWA across different workloads, with $\epsilon = 0.1$.

Fig. 6 shows results for four different workloads and three different datasets at $\epsilon = 0.1$. The datasets span the range of difficulty from the "easier" Nettrace to the "harder" HepPh. (The algorithms being compared here are affected by the dataset because they operate on the data-dependent partition selected in the first stage.) The original DAWA performs very well on all cases. In particular, it always outperforms Partition+Geometric on *uniform interval* workload and performs exactly same as the Partition+Laplace mechanism on *identity* workload. In the latter case, we find that the greedy algorithm in the second step outputs the initial budget allocation, which is exactly same as the Laplace mechanism.

5.4 Case study: spatial data workloads

Lastly, we evaluate an extension to our main algorithm to compute histograms over two dimensional spatial data. We use an experimental setup that is almost identical to previous work [6]; differences are highlighted below. The dataset describes the geographic coordinates (latitude and longitude) of road intersections, which serve as a proxy for human population, across a wide region in the western U.S. [2]. Over this region, we generate a workload of random rectangle queries of four different shapes: $(1, 1)$, $(5, 5)$, $(10, 10)$, and $(15, 0.2)$ where shape (x, y) is a rectangle that covers x degrees of longitude and y degrees of latitude.

We compare with a data-independent algorithm, QuadTree [6], and data-dependent algorithms MWEM, HybridTree [6], and DPCube [21]. Among these algorithms, only MWEM is workload-aware. Since some algorithms expect discrete domains as input, we discretize the domain by partitioning the space into the finest granularity used by the QuadTree, whose height is 10 [6]. Thus, both longitude and latitude are split evenly into 2^{10} bins.

To extend the DAWA algorithm to two dimensional data, we use a Hilbert curve of order 20 to convert the $2^{10} \times 2^{10}$ grid into a 1-dimensional domain with size 2^{20}. In case the query region only partially covers some bins in the discretized domain, the query answer is estimated by assuming uniformity within each bin.

Fig. 7 shows the results. Although DAWA is designed for interval workloads on 1-dimensional data, it performs as well or better than algorithms specifically designed to support rectangular range queries on 2-dimensional data. The performance gap between DAWA and its competitors increases as ϵ decreases.

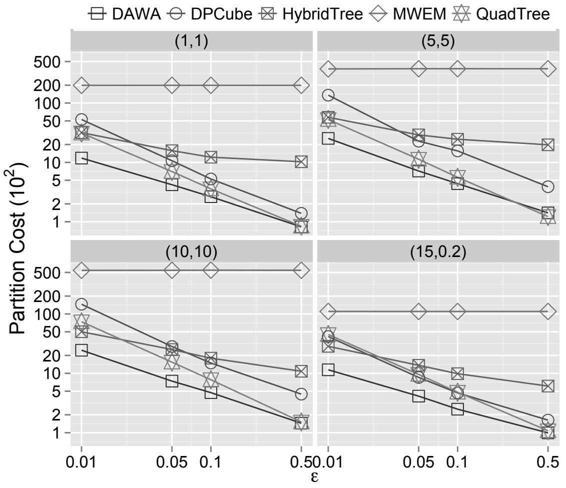


Figure 7: Average error answering query workloads on spatial data. Each workload is a batch of random rectangle queries of a given (x, y) shape.

6. RELATED WORK

A number of data-dependent algorithms have been developed recently [3, 6, 12, 21, 22]. We empirically compare DAWA against these approaches in Sec. 5. Our algorithm is most similar to P-HP [3] and Structure First [22], both of which find a partition and then compute statistics for each bucket in the partition. P-HP's approach to partitioning is based on the same optimization as presented here (Problem 1). It uses the exponential mechanism to recursively bisect each interval into subintervals. A key distinction is that P-HP is an approximation algorithm: even if $\epsilon \to \infty$, it may not return the least cost partition. In contrast, we show that the optimization problem can be solved directly by simply using noisy scores in place of actual scores and we prove that a constant amount of noise is sufficient to ensure privacy. The experiments in Sec. 5.3.2 show that P-HP consistently returns higher cost partitions than our approach. Structure First [22] aims to solves a different optimization problem (with a cost function based on L_2 rather L_1). In addition, it requires that the user specify k, the number of buckets, whereas DAWA automatically selects the best k for the given dataset. Neither P-HP nor StructureFirst is workload-aware.

The other data-dependent mechanisms use a variety of different strategies. The EFPA [3] algorithm transforms the dataset to the Fourier domain, samples noisy coefficients, and then transforms back. DPCube [21] and Hybrid Tree [6], both of which are designed for multi-dimensional data, construct estimates of the dataset by building differentially private KD-trees. MWEM [12] derives estimates of the dataset iteratively: each iteration selects a workload query, using the exponential mechanism, and then updates its estimate by applying multiplicative weights given a noisy answer to the query. MWEM supports the more general class of linear queries, whereas DAWA is designed to support range queries. MWEM also offers strong asymptotic performance guarantees. However, on workloads of range queries, we find in practice that MWEM performs poorly except when the dataset is highly uniform. It is also limited by the fact that it can only ask workload queries, which may not be the best observations to take.

General data-dependent mechanisms are proposed in the theory community [10, 11]. They are not directly comparable because they work on a slightly weaker variant of differential privacy and are not computationally efficient enough to be practical.

Data-independent mechanisms attempt to find a better set of measurements in support of a given workload, then apply the Laplace mechanism and inference to derive consistent estimates of the workload queries. Our DAWA algorithm would be similar to these methods if the partitioning step always returned the trivial partition, which is $\mathbf{x}$ itself. Many of these techniques fall within the matrix mechanism framework [15], which formalizes the measurement selection problem as a rank-constrained SDP. While the general problem has high computational complexity, effective solutions for special cases have been developed. To support range queries, several mechanisms employ a hierarchical strategy [6, 13, 20]. Our approach builds on this prior work. A key difference is that our algorithm adapts the strategy to fit the specific set of range queries given as a workload, resulting in lower workload error. Other strategies have been developed for marginal queries [5, 7]. Yuan et al. [24] revisit the general problem for the case when workloads are small relative to the domain size; however the algorithm is too inefficient for the domain sizes we consider here. Other algorithms have been developed that adapt to the workload. However, they are not directly applicable because they are designed for a weaker variant of differential privacy [16], or employ a user-specified "recovery" matrix, instead of ordinary least squares [23].

7. CONCLUSION & FUTURE WORK

DAWA is a two-stage, data- and workload-aware mechanism for answering sets of range queries under differential privacy. DAWA first partitions the domain into approximately uniform regions and then derives a count for each region using measurements of varying accuracy that are tuned to the workload queries. Experimentally, DAWA achieves much lower error than existing data-dependent mechanisms on datasets where data-dependence really helps. On complex datasets, where competing data-dependent techniques suffer, DAWA does about the same or better than data-independent algorithms. In this sense, DAWA achieves the best of both worlds.

Our results have shown that, for some datasets, data-aware algorithms can reduce error by a factor of 10 or more over competitive data-independent techniques. But it remains difficult to characterize exactly the properties of a dataset that permit lower error under differential privacy. Optimal partition cost of a dataset provides some insight into dataset "hardness" for the DAWA algorithm, but we are not aware of a general and fully-satisfying measure of dataset complexity. We view this as an important direction for future work. We would also like to consider extensions to private partitioning that would directly incorporate knowledge of the workload and to extend our method to a larger class of worloads beyond one- and two-dimensional range queries.

Acknowledgements. We are grateful for the comments of the anonymous reviewers. This work is supported by the NSF through grants CNS-1012748, IIS-0964094, CNS-1129454, and by CRA/-CCC through CI Fellows grant 1019343.

8. REFERENCES

[1] http://snap.stanford.edu.

[2] http://www.census.gov/geo/maps-data/data/tiger.html.

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

[4] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[5] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *PODS*, pages 273 – 282, 2007.

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

[7] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, pages 217–228, 2011.

[8] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[9] C. Dwork, F. M. K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[10] C. Dwork, G. N. Rothblum, and S. P. Vadhan. Boosting and differential privacy. In *FOCS*, pages 51–60, 2010.

[11] A. Gupta, A. Roth, and J. Ullman. Iterative constructions and private data release. In *TCC*, pages 339–356, 2012.

[12] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, pages 2348–2356, 2012.

[13] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1-2):1021–1032, 2010.

[14] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

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

[16] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012.

[17] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30, 2009.

[18] S. Ruggles, J. Alexander, K. Genadek, R. Goeken, M. Schroeder, and M. Sobek. Integrated public use microdata series: Version 5.0, 2010.

[19] United States Department of Health, Human Services. Centers for Disease Control, and Prevention. National Center for Health Statistics. National home and hospice care survey, 2007.

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

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

[22] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, pages 1–26, 2013.

[23] G. Yaroslavtsev, G. Cormode, C. M. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. In *ICDE*, 2013.

[24] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: Optimizing batch queries under differential privacy. *PVLDB*, 5(11):1136–1147, 2012.