Automatic, Fine-Grained Algorithmic Choice for Differential Privacy

Jacob Imola
School of Computer Science, Carnegie Mellon University

Jean Yang
School of Computer Science, Carnegie Mellon University

April 23, 2018

Chapter 1

Introduction

The rapid technological increase in data collection, speed, and storage has brought about revolutionary insights and ideas and will continue to do so. However, with huge amounts of private data comes the concern of preventing data from ending up in the wrong hands. In order to prevent data leakage, we must lay a strong privacy foundation and give data programmers tools for implementing privacy both efficiently and correctly.

Consider a healthcare database with records like patient weight, age, genetic information, and whether they are HIV positive. Granting access rights, or policies, to just patients and their doctors protects as much privacy as possible, and developing tools for verifying information flow policies is an interesting question in its own right that has been studied immensely. However, sometimes it's okay to release some statistics about the database so that a programmer can find risk factors for people who have HIV. Publicly releasing the entire database doesn't protect privacy at all, yet it would be a programmer's dream. In order to appease both data programmers and patients, a middle ground area must be chosen where a blurry snapshot of the database is released, comprehensive enough so that meaningful conclusions may be drawn yet blurry enough so that individuals are mostly protected. We call this the *privacy-accuracy* tradeoff. We can always sacrifice one for the other before we disclose our database snapshot. However, after we disclose, it is impossible to take any privacy back, so we have to be absolutely sure that privacy guarantee will not fail under any attack. The most promising method for doing such a disclosure is Differential Privacy [4].

Differential Privacy (DP) is considered to be the gold-standard of privacy

and has been researched intensely since its conception in 2005. Its goal is to provide guarantees on what can be done with the information being released from a dataset while making minimal assumptions about an attacker's abilities. Previous attempts at privacy were susceptible to surprise exploits that occurred after a database was released. Notably, before DP, researchers were able to reidentify users in a Netflix dataset given an auxiliary dataset from IMDB and form a generalized attack against the state-of-the-art privacy algorithms of the time [18]. The strong privacy guarantee of DP, on the other hand, has a rigorous mathematical foundation that makes it impervious to the post-processing attacks that compromised the Netflix dataset, and more recently, AirBnB and Instagram. Differential Privacy has stood the test of time as a sturdy way to protect privacy.

However, just building a suite of DP algorithms is not satisfactory. Differential Privacy necessarily adds randomness to programs, and randomness adds a new layer of complexity to programs. For example, adding noise to a variable that governs how many times a loop is performed could result in two highly different program executions and thus a very noisy answer. However, in many DP applications, noise can be added in a number of places, different queries could be done to accomplish the same goal, and so on. A very noisy answer could potentially be avoided if the proper algorithm is deployed. If one algorithm always dominated the others, this wouldn't be much of a problem, but as we will see, the best one depends on the database in a complex way. Picking the best algorithm for the job depends on extensive empirical evaluations, as theoretical bounds are often insufficient for complex algorithms [8]. This leads the central problem addressed by this work:

Problem: How can we alleviate the burden of empirical algorithmic choice from the programmer?

This problem is noted by the authors of DPComp [8], who comment that currently, "the practitioner is lost and incapable of deploying the right algorithm". DPComp allows the programmer to visualize the performances of different histogram querying algorithms on certain datasets. An example of one algorithm executed on one dataset is shown in Figure 1.1. We feel this is a step in the right direction, but their approach would be better if its ideas were applied to a programming language. The approach is to move the manual exploration of algorithms that a programmer might do with DPComp into the runtime of our programming language, which we call `Jostle`, so named because it will "jostle" the various algorithms during runtime. This

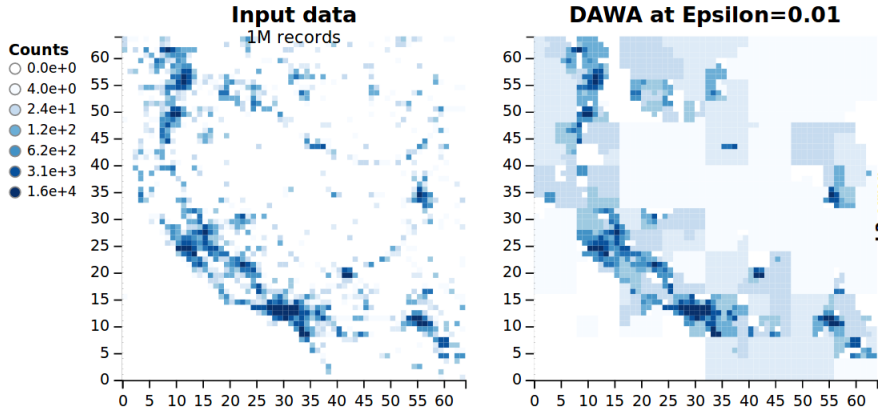

Figure 1.1: A screenshot of a DPComp graph depicting the original dataset and the noise that a DP algorithm, DAWA [13], adds at $\epsilon = 0.01$. This would help a programmer decide whether to apply DAWA on his own dataset.

will give the following benefits:

1. **Correctness** A `Jostle` program enjoys the correctness guarantees that existing DP programming languages give. Correctness in the area of privacy is particularly important because privacy breaches are costly.

2. **Generalization** The DPComp study of 2D histograms does not easily generalize to other DP algorithms. A programming language is far more general as arbitrary code can be analyzed. Concretely, DPComp can be implemented as a specific `Jostle` program, and not the other way round.

3. **Differential Privacy Insight** DPComp is also a teaching tool because it advances one's knowledge of how algorithms perform on datasets. Every time a `Jostle` program picks a algorithm, its trace extends a programmer's understanding of performance improves. This insight is particularly helpful to programmers new to DP who have no clues as to which algorithm to deploy.

Even though programming languages exist which address benefit Item 1 and DPComp addresses Items 2 and 3, no works enjoy all three benefits at the same time. Having all three benefits is important because of programmer burden. Doing a DPComp-style analysis every time algorithmic choice needs

```
1  trainingSet = {US Census, ... #Public Databases}
2  def uniformity(db):
3      xgrp = sqrt(db.x_range)
4      ygrp = sqrt(db.y_range)
5      return {{stddev(select sum(values) from db
6                      groupby x/xgrp, y/ygrp)}}
7  def sparsity(db):
8      num_nonzero = {{select count(values) from db
9                      where values > 0}}
10     domain_size = db.x_range * db.y_range
11     return num_nonzero/domain_size
12
13 #Option
14 def DAWA(db, queries):
15     #DAWA Implementation
16 #Option
17 def MWEM(db, queries):
18     #MWEM Implementation
19
20 noisyHistChoice = MkChoiceMaker among {DAWA, MWEM}
21                      informed by {uniformity, sparsity}
22                      trained on trainingSet
23
24 def answerHistQueries(db, queries):
25     answers = noisyHistChoice(db, queries)
```

Figure 1.2: `Jostle` code demonstrating the use of `ChoiceMaker` object, called `noisyHistChoice`. In this case, the options are the `DAWA` and `MWEM` 2D Histogram algorithms.

to be explored requires doing data analysis on all algorithms and then implementing the insights by hand. Once algorithms are implemented, there is a small overhead writing `Jostle` code to make the choice. The algorithmic insights come for free via the `Jostle` trace.

`Jostle` will represent algorithmic choice with the `MakeChoice` statement, illustrated in Figure 1.2. A programmer will use a `ChoiceMaker` when they are unsure of which choice to make at a certain point in their code; in the example, they are choosing between the algorithms `DAWA` and `MWEM`. The constructor, `MkChoiceMaker`, takes in a set of `Options`, a set of training databases, and a set of functions from the database to $\mathbb{R}$ called metafeatures.

The above design is inspired by DPComp. Suppose we are a programmer deciding whether to run the `DAWA` algorithm on a public dataset shown in Fig-

ure 1.1. We may notice that the input is a rather sparse and non-uniform—most of the points are 0 (white) and the non-zero points are clustered into groups. Indeed, this dataset is the population distribution for the southwest United States. We decide that these two properties adversely affect the performance for `DAWA`, as it predicts many of the white points incorrectly to have a nonzero value. Similarly, `sparsity` might affect the performance of `MWEM`, but not `uniformity`. The set of all properties that may impact performance are the *metafeatures*. They are database properties that we pay attention to when we estimate algorithm performance.

`MkChoiceMaker` will train a model from the set of metafeatures to predicted algorithm performance. The resulting `ChoiceMaker` can be called on a private database. When this happens, the metafeatures are evaluated on the database and the `Option` with the highest estimated performance is dispatched.

We will begin with a background section on DP and related work. Then, we will present a formal outline of how `Jostle` runs. Thirdly, we will implement Decision Tree choice with `Jostle` and provide empirical evidence for the claimed benefits (Page 3) of `Jostle`. Finally, we will discuss future directions for `Jostle`.

Chapter 2

Background

2.1 Differential Privacy

Differential Privacy makes the following promise to data subjects: "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available" [4]. To make this more formal, we must analyze two databases, D and D', differing in only one row which represents a database before and after a user participates. We call such databases neighbors. Suppose we are running a algorithm $\mathcal{M}$ on the database. If an attacker is able to discern with confidence $\mathcal{M}(D)$ and $\mathcal{M}(D')$, then this poses a privacy threat. The strength of this confidence is quantified a real number ϵ such that small ϵ corresponds to low attacker confidence. This means that deterministic algorithms are already unacceptable if it's possible for $\mathcal{M}(D) \neq \mathcal{M}(D')$. We necessarily must output a probability distribution, and once we view $\mathcal{M})(D)$ as a distribution, we can finally pin down the definition:

Definition 1. *$\mathcal{A}$ satisfies ϵ-DP if for all D and D' such that $|D - D'|_1 = 1$ and for all o in the range of $\mathcal{M}$,*

$$\Pr\left(\mathcal{M}(D) = o\right) \leq e^{\epsilon} \Pr\left(\mathcal{M}(D') = o\right)$$

There is also a more general definition that gives a weaker privacy guarantee:

Definition 2. *$\mathcal{M}$ satisfies (ϵ, δ)-DP if for all D and D' such that $|D - D'|_1 = 1$ and for all o in the range of $\mathcal{M}$,*

$$\Pr\left(\mathcal{A}(D) = o\right) \leq e^{\epsilon} \Pr\left(\mathcal{M}(D') = o\right) + \delta$$

For much of this paper, we will focus on ϵ-DP, but it is worth knowing the more general case so we can import the well-known privacy theorems in their full generality.

The definition doesn't address why the "no matter what" part of the promise is true, but we can view any post-release attack on $\mathcal{A}$ as a function F that doesn't involve D. Then, the following theorem establishes the promise:
Theorem 1. *(Post-Processing [4]) If $\mathcal{M}$ satisfies (ϵ, δ)-DP, and F is any function that takes the output of $\mathcal{M}$ as input, then $F(\mathcal{M})$ satisfies (ϵ, δ)-DP.*

This theorem is the reason why DP is such a useful guarantee. Data programmers can be sure that once they run their algorithm $\mathcal{M}$ and release its output, then the DP guarantee gets no weaker *no matter what an adversary does with the data.* This prevents the headaches where a programmer realizes retroactively that the data he released can be combined in some way to reveal much more information than was intended, like in Netflix [18].

In addition, DP satisfies several other useful properties:
Theorem 2. *(Composition [4]) Given algorithm M_1 and M_2 satisfying ϵ_1 and ϵ_2 DP, respectively, along with a database D, the algorithm $M = (M_1(D), M_2(D))$ has $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$ DP.*

Composition is like the union bound from probability; it's convenient to apply but often is a pessimistic bound, as we will see later. Because of composition, we often refer to ϵ as a privacy budget—if we string together many private computations, it's like we spend some of our budget on each one out of a total budget of ϵ.

We can easily improve upon Composition in the special case of algorithms operating on disjoint parts of the database. If D is split into disjoint parts before algorithms are applied to it, then out of all its possible neighbors, only one of the parts will be different. Thus, only the worst algorithm will affect the DP guarantee:
Theorem 3. *(Disjointness [4]) Given disjoint subsets D_1, D_2 of D with two algorithms M_1 and M_2 providing (ϵ_1, δ_1) and (ϵ_2, δ_2)-privacy, then $((M_1(D_1), M_2(D_2))$ satisfies $(\max\{\epsilon_1, \epsilon_2\}, \max\{\delta_1, \delta_2\})$-DP.*

So, what's a simple example of a DP algorithm? Suppose each row of our database D is 0 or 1, so $D \in \{0,1\}^n$, and that we are trying to release the sum of the elements of D. If this sum is S, then all neighboring databases D' have sum S or $S+1$. We can add noise to S so that it looks very similar in distribution to $S+1$. The distribution we are looking for is the Laplace distribution:

Definition 3. *The Laplace(λ) distribution has probability mass function* $f(x) = \frac{1}{2\lambda} e^{-|x|/\lambda}$.

This distribution fits perfectly with the definition of DP because of the exponentials. If X, Y are i.i.d. from Laplace $\left(\frac{1}{\epsilon}\right)$, then it is straightforward to show that the distributions of $S + X$ and $S + 1 + X$ satisfy $(\epsilon, 0)$ DP. To generalize this statement, we will use the following definition:

Definition 4. *(Sensitivity) A function f is Δ-sensitive if for all x, y such that $|x - y|_1 = 1$, we have*

$$|f(x) - f(y)| \leq \Delta$$

This can equivalently be rephrased as

$$\max_{|x-y|_1=1} |f(x) - f(y)| = \Delta$$

We will denote the sensitivity of f by $\Delta(f)$.

This gives us the following algorithm:

Algorithm 1: Laplace algorithm

Input : D, a database; f, a function $\mathcal{D} \to \mathbb{R}^n$; and ϵ

Output: An estimate for $f(D)$ satisfying ϵ-DP.

1 X, a vector of n i.i.d. variables drawn from Laplace $\left(\frac{\Delta(f)}{\epsilon}\right)$;

2 **return** *X+f(D)*

Theorem 4. *The Laplace algorithm 1 satisfies $(\epsilon, 0)$-DP [4].*

For the counting or histogram queries such as our example above, we have $\Delta = 1$ so we add Laplace $\left(\frac{1}{\epsilon}\right)$ noise to our function.

However, what if we wanted to compute the maximum value in a set? If we had n elements, we certainly wouldn't want to apply Composition n times, obtaining $n\epsilon$-DP, just to have `Laplace` $\left(\frac{1}{\epsilon}\right)$ noise added to our answer. A better way is to use ReportNoisyMax 2. Instead of paying $n\epsilon$, ReportNoisyMax allows us to pay ϵ for the exact same noise on our answer. However, ReportNoisyMax only works on monotone queries, or where $f(x, D) < f(x, D')$ for all $x \in X'$. A version that works on queries in general is the exponential mechanism 3. ReportNoisyMax doesn't have a factor of 2 in its Laplacian noise, and this means a lighter tail. Thus, for monotone queries, we use ReportNoisyMax.

Algorithm 2: ReportNoisyMax

Input : $D \in \mathcal{D}$, $\mathcal{X}$, a domain; f, a function $\mathcal{X} \times \mathcal{D} \to \mathbb{R}$; and ϵ

Output: $x \in \mathcal{X}$ that attains maximum value on $f(S)$, satisfying ϵ-DP.

1 X, a vector of $|\mathcal{X}|$ i.i.d. variables drawn from Laplace $\left(\frac{\Delta(f)}{\epsilon}\right)$;

2 **return** $\arg\max_{i=1}^{|\mathcal{X}|}\{X + f(\mathcal{X})\}$

Algorithm 3: exponential mechanism

Input : $D \in \mathcal{D}$; $\mathcal{X}$, a domain; $f : \mathcal{X} \times \mathcal{D} \to \mathbb{R}$, a utility function, ϵ

Output: $x \in \mathcal{X}$ where $f(x, D)$ is more likely to be high.

1 Pick $x \in \mathcal{X}$ where $\Pr(x = k) \propto \exp\left(\frac{\epsilon f(k,D)}{2\Delta(f)}\right)$;

2 **return** x

2.2 Related Work

2.2.1 Existing Programming Languages

Existing programming languages for differential privacy leave algorithmic choice up to the programmer. The most prominent example for a language with a DP runtime is PINQ [16]. PINQ provides an interface of DP primitives between the database and the programmer. These primitives include aggregations, database splitting, and the exponential mechanism. PINQ uses the Laplace algorithm to compute the aggregations. To string together many commands, PINQ gives the user the ability to define a PINQAgent which keeps track of budgets in a compositional way. For example, the `NoisyCount` function is implemented in Figure 2.1 PINQ includes the most basic privacy primitives for simplicity, and lots of work has gone into includ-

```
double NoisyCount(double epsilon){
    if(myagent.apply(epsilon)){
        return mysource.Count() + Laplace(1.0/epsilon);
    }else{
        throw new Exception("Access Denied")
    }
}
```

Figure 2.1: NoisyCount Implemented in PINQ.

ing more sophisticated algorithms into its runtime such as `Join` [19] [11]. However, in the absence of automatic algorithmic choice, a programmer will not know

In Fuzz [21], a type system is implemented to guarantee differential privacy and sensitivity. Each type is endowed with a metric, and judgments are given for richer programming constructs such as sums, products, recursive types, and lambda expressions. Once the type of the program is known, its metric is known and its sensitivity can be inferred from the input variables. Sensitivity allows us to add the proper amount of noise via the Laplace algorithm. To go from an ϵ-sensitive function to an ϵ-DP function, noise is added via a monad by applying the function $add\_noise : \mathbb{R} \multimap \bigcirc\mathbb{R}$.

For instance, counting the number of people in a database older than 40 could be achieved by:

$$add\_noise\ (size\ (filter\ over_{40}\ db)) : db \multimap \bigcirc\mathbb{R}$$

However, there are two shortcomings with this method. First, the programmer is still responsible for adding noise and making decisions that affect the sensitivity of a computation. Perhaps it's better to defer the adding of noise of the

2.2.2 Existing Methods for Algorithmic Choice

Versions of algorithmic choice have been demonstrated in previous work. Each work provides a way of overcoming the fact that the choice of algorithm releases information about that dataset in a way that is more intelligent than simply using composition for each algorithm. We set up the the problem in the following way: Given some model space $\mathcal{M}$, a model set $M \subseteq \mathcal{M}$, a database D, and a score function $q : \mathcal{R} \times \mathcal{M} \to \mathbb{R}$, maximize q over $\mathcal{M}$ privately.

For instance, if the programmer is training a private linear regression model, then $\mathcal{M}$ is the space of all linear functions and q is, most commonly, a validation score on some unused part of the database. This problem is extra subtle as changing D to D' changes the model set from M to M', as the models are determined on the database, and also changes D to D' when evaluating $q(D, m)$. There is no complete answer to this problem; most commonly, an assumption must be made about q, usually relating to its sensitivity. We now explore proposed solutions to this problem.

In [3], a solution based on the exponential mechanism is proposed where q is the utility function. Since the exponential mechanism requires a sensitivity bound, a bound on q must be ascertained. As mentioned, in order to bound $|q(D,m) - q(D',m')|$ where $m \in M$ and $m' \in M'$, two inequalities are needed:

$$\forall m \in M, \forall |D - D'| = 1 : \ \|q(D,m) - q(D',m)\| \leq \beta_1$$
$$\forall D \in \mathcal{R}, \forall m \in M, \forall m' \in M' : \ \|q(D,m) - q(D,m')\| \leq \beta_2$$

If these bounds are established, then it's relatively easy to show that the exponential mechanism can output an answer that is close to optimal (compare to the guarantee of the exponential mechanism):

Theorem 5. *(Utility guarantee [3]) Let $M = m_1, m_2, \ldots, m_k$. Then, with probability at least $1-\delta$, $q(D, m_{i^*}) \geq \max_{1 \leq i \leq k} q(D, h_i) - \frac{2 \max\{\beta_1, \beta_2\} \log(k/\delta)}{\epsilon}$.*

We can see that the bound degrades logarithmically in the size k of the universe. A fix to this was made in [2]. Suppose there is a relatively small subset of models $M' \subseteq M$ which perform much better than the rest of M, so $\min_{m \in M'} q(D,m') \geq \max_{m \in M} q(D,m) + \delta$. It's possible to find such an M' with the sparse vector technique. With a smaller subset of better-performing algorithms, the exponential mechanism works will produce a much better result.

However, neither paper gives insight into how to choose ϵ. How much ϵ should we use for the exponential mechanism, and how much should be used for actually computing the models? How much ϵ should be given to the sparse vector computation before we even do the exponential mechanism? Different frameworks are needed to eliminate the need to pick ϵ.

The framework developed in [14] allows the programmer to experimentally pick ϵ in certain cases. Critical to the framework is the method of picking correlated Laplacian noise described in [12]. In this version of the Laplace algorithm, a programmer selects a set of increasing ϵ values, $(\epsilon_1, \epsilon_2, \ldots, \epsilon_T)$, corresponding to the different budgets they want to try. Correlated Laplace variables $(v_1, v_2, \ldots, v_T)$ are then generated such that knowing a prefix $(v_1, v_2, \ldots, v_t)$ is ϵ_t-differentially private and the noise present in v_t is similar to the noise that a standard Laplace algorithm would add. The framework in [14] uses this algorithm to add noise to models $m_1, m_2, \ldots, m_T$ when the sensitivities of the models are known. For instance, if the task is linear regression, then the describing a way to iterate through the v_i's until a suitably

accurate algorithm is selected. Accuracy can be specified by the programmer as an arbitrary function that takes in a v_i. Of course, the release of the most-accurate algorithm has to be done in a differentially-private manner as well, and Ligett et. al use an adaptation of the AboveThreshold algorithm [4]. This results in an additional privacy and accuracy penalty over simply using the Laplace algorithm with a fixed ϵ. Specifically, if v_t is chosen, then the computation is $\epsilon_t + \epsilon_{Above}$-DP, still a huge improvement over the bound obtained when applying $\sum_{i=1}^{t} \epsilon_i$ with nearly the same accuracy.

We notice the following problems with the methods addressed above:

- AboveThreshold does not work on releasing (ϵ, δ)-DP algorithms with $\delta > 0$.
- A privacy and accuracy penalty is taken when selecting the best algorithm. This method is rather arbitrary; how do we select ϵ_A? Is this even the optimal way of algorithm selection?
- The papers only work on the specialized case of the Laplace algorithm.

  More papers to talk about: [23], [15], [9].

Chapter 3

Experiments and Implementation

3.1 Decision Trees

Decision Trees are a powerful tool for data mining due to their high human interpretability, non-parametric design, low computational cost, ability to discover non-linear relationships among attributes, resilience to missing values, ability to handle both discrete and continuous data, and ability to handle non-binary labels [6]. Let our database again be D, and suppose it has k columns or attributes, the ith of which can take values in $\mathcal{A}_i$. Let $\mathcal{C}$ be the output attribute, or class, which we are trying to predict. We assume for simplicity that $\mathcal{A}_i$ and $\mathcal{C}$ are discrete sets. A decision tree classifies points by branching on attribute $\mathcal{A}_i$, forming $|\mathcal{A}_i|$ subtrees. Once certain criteria are met, no more branching occurs, and instead a leaf node predicts the class. An example Decision Tree is given in Figure 2.2.

The most widely-used algorithm for training decision trees is the C4.5 algorithm. C4.5 grows trees top-down, and it creates a branch up to a certain depth specified by the user. For each branch, it selects the attribute that produces the lowest conditional entropy and splits the dataset on it. Conditional Entropy is defined as:

Definition 5. *(Entropy) The Entropy of a discrete random variable X*

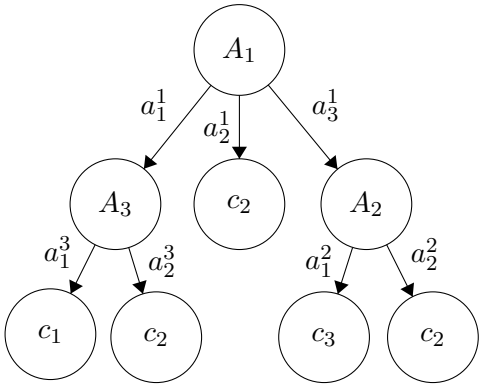

Figure 3.1: Example Decision Tree.

which attains k values with probabilities $p_1, p_2, \ldots, p_k$ is

$$H(X) = -\sum_{i=1}^{k} p_i \log(p_i)$$

The conditional entropy of X given a discrete random variable Y which attains values $a_1, \ldots, a_\ell$ is

$$H(X \mid Y) = \sum_{i=1}^{\ell} \Pr[Y = i] H(X \mid Y = i)$$

Conditional entropy, being a measure for information, is minimized so as to prioritize those attributes which produce large information gain. For leaf nodes, the class that has the largest representation in the remaining dataset is selected. The C4.5 algorithm appears in algorithm 2.3. We denote by $D_x^{(i)}$ to be the subset of D which attains value j on attribute i and $D_{x,y}^{(i)}$ to be the subset of $D_x^{(i)}$ which also has class y. Let's let $\tau_{x,y}^{(i)}$ be the size of $D_{x,y}^{(i)}$. In the code, we represent this as `D[i=x]` and `D[class=y]`, respectively. Defining conditional entropy in our new notation, we get, for attribute with index i:

$$H_i(D) = \sum_{j \in \mathcal{A}_i} \frac{\tau_j^{(i)}}{\tau} \sum_{c \in \mathcal{C}} \frac{\tau_{j,c}^{(i)}}{\tau_j^{(i)}} \ln\left(\frac{\tau_j^{(i)}}{\tau_{j,c}^{(i)}}\right) \to \sum_{j \in \mathcal{A}_i} \sum_{c \in \mathcal{C}} \tau_{j,c}^{(i)} \ln\left(\frac{\tau_j^{(i)}}{\tau_{j,c}^{(i)}}\right) \tag{3.1}$$

We omit a $\frac{1}{|D|}$ factor on the right side as we often simply compare computations on a fixed D. Two other estimates for the quality of a split which

```
1  def dtree(D, atts, clss, d):
2    if(d = 0 or len(atts) = 0):
3      best_class = max over c in clss of len(D[clss=c, ])
4      return Leaf(pred=best_class)
5    else:
6      best_att = max over A in atts of:
7        c_entropy(D, A, clss)
8      C = map(best_att, lambda a: dtree(D[best_att=a, ],
9        att-best_att, clss, d-1))
10     return Node(att=best_att, children=C)
```

Figure 3.2: C4.5 algorithm.

are generally less good in the non-private setting are Gini and Max:

$$G_i(D) = \sum_{j \in A_i} \tau_j^{(i)} \left(1 - \sum_{c \in C} \left(\frac{\tau_{j,c}^{(i)}}{\tau_j^{(i)}} \right)^2 \right) \tag{3.2}$$

$$M_i(D) = \sum_{j \in A_i} \max_c (\tau_{j,c}^{(i)}) \tag{3.3}$$

When converting this algorithm to the differentially-private version, the user is left with several questions as noted in [6]:

- How large of a budget has been alotted or should be alotted? There isn't a clear way to decide the budget, as the performance of the algorithm may vary wildly with the budget.
- How many times should the data be queried? How would pick d in C4.5? Should one alter line (1) to something different?
- Might the sensitivity of some of the queries prevent an accurate choice? Specifically, even though it is widely agreed that the entropy function performs best in the non-private setting, could a lower-quality function be substituted because its computation is more accurate?
- How does the size of D impact performance? Is there enough data to provide accurate results in the private setting?

As we will see, the answers to these questions are data-dependent and the there is never one answer that always dominates.

Chapter 4

Solution Overview

Here, we describe in detail how `Jostle` is implemented.

Name	Type
Database Space	$\mathcal{R}$
Metafeature Space	$\mathcal{X}$
Query Answer Space	$\mathcal{Q}$
`answer`	$\mathcal{R} \to \mathcal{Q}$
`predict`	$\mathcal{X} \to \mathbb{R}$
`Option`	$(\mathcal{R} \to \mathcal{Q}) \times (\mathcal{X} \to \mathbb{R})$
`MkChoiceMaker`	$(\texttt{Option list} \times \mathcal{R}\ \texttt{list}) \to (\mathcal{R} \to \mathcal{Q})$
`ChoiceMaker`	$\mathcal{R} \to \mathcal{Q}$

- `answer` The implementation of the algorithm. Called once the choice has been made.
- `predict` A prediction made by the programmer and by `Jostle` as to how well an algorithm
- `Option`
- `MkChoiceMaker`
- `ChoiceMaker`

4.0.1 Private Decision Trees

We assume, as does most of the literature, that the $\mathcal{A}_i$ and $\mathcal{C}$ are public and D is private. Also, denote our total privacy budget to be β. For recursive decision tree algorithms set up like 2.3, the recursive calls on the same depth of the tree all operate on disjoint subsets of D. Therefore, a conservative estimate of the privacy usage, using composition and disjointness, is

$$\sum_{i=0}^{d} \max_{n \in \texttt{Nodes on lvl } i} \epsilon_n \tag{4.1}$$

where ϵ_n is the privacy used by node n. We will go into details on how to pick ϵ_n later.

The most naive way to create a DP version of C4.5 is to take all the histogram queries in algorithm 2.3 and change them to `NoisyCount`. This amounts to computing the dataset size, the most common class of a leaf, and the conditional entropy 2.1 of a branch. Due to the disjointness of the $D_{j,c}^{(i)}$ with each other and the $D_j^{(i)}$ with each other with a fixed i, we can do a noisy count on each $|D_j^{(i)}|$ and $|D_{j,c}^{(i)}|$ with $\frac{\epsilon'}{2}$ budget (sensitivity 1), and compute the conditional entropy spending just ϵ' budget. Unfortunately, doing this over all attributes is not disjoint, so we have to split our ϵ_n budget over up to k attributes and use composition. This adds a lot of noise to our computations, namely each NoisyCount gets $\frac{\beta}{dk}$ budget. This method is presented in [1] as a proof of concept rather than a high-performing algorithm.

To fix this accuracy problem, Friedman and Schuster [7] use entropy as a utility function on a call to the exponential mechanism. Specifically, they operate over the domain $\{1, 2, \ldots, k\}$, the indices of the attributes, and utility function $u(D, i) = -H_i(D)$, where the entropy is negated because we want to find the attribute with greatest entropy reduction. They also try functions other than entropy as a quality estimator because entropy has a rather high sensitivity, shown in Theorem 6.

Theorem 6. *The entropy function on disjoint histogram counts $a_1, a_2, \ldots, a_n$ produced from a database D has sensitivity bounded by $\frac{1}{|D|}\left(\frac{1}{\ln(2)} + \log(|D|)\right)$.*

Proof. Let $A = \sum_{i=1}^{n} a_i$. Then, the entropy is

$$\sum_{i=1}^{n} \frac{a_i}{A} \log\left(\frac{A}{a_i}\right) = \frac{1}{A}\sum_{i=1}^{n} a_i \log A - \frac{1}{A}\sum_{i=1}^{n} a_i \log(a_i) = \log(A) - \frac{1}{A}\sum_{i=1}^{n} a_i \log(a_i)$$

Suppose bucket a_j is reduced by 1, and the entropy change is

$$\log(A) - \log(A-1) - \frac{1}{A}a_j \log(a_j) + \frac{1}{A-1}(a_j - 1)\log(a_j - 1)$$
$$\leq \frac{1}{\ln(2)(A-1)} - \frac{1}{A}(a_j - 1)\log(a_j - 1) + \frac{1}{A-1}(a_j - 1)\log(a_j - 1)$$
$$= \frac{1}{\ln(2)(A-1)} + \frac{1}{A(A-1)}(a_j - 1)\log(a_j - 1) \leq \frac{1}{\ln(2)(A-1)} + \frac{1}{A}\log(A)$$

□

Another big change is the stopping criteria (Line 2 in Figure 2.3). As noted in [6], the stopping criteria could be different in differential privacy versus the regular setting because Laplace $\left(\frac{1}{\epsilon}\right)$ creates much higher error in smaller leaves, and the depth of a tree affects the amount noise added. On a high level, this suggests that shorter, sparser trees will perform better in the DP setting, but the ultimate relationship remains unclear. In [7], an additional stopping parameter depending on the NoisySize of D and some other public parameters are used in addition to the stopping parameters in C4.5. Their goal is to ensure that a certain signal is larger than the noise of NoisyCount:

$$\frac{|D|}{t|C|} < \frac{\sqrt{2}}{\epsilon}$$

where $\frac{|D|}{t|C|}$ is a signal for how large the partitions that the children nodes will make. The sizes of D for each node is used again when doing the standard C4.5Prune algorithm [20] which doesn't use a validation set but estimates a confidence interval from the results of the training set. To do this, it needs an estimate on the sizes of $D[class = c]$ for each $C \in \mathcal{C}$ at each node, and knowing the size of D helps Friedman and Schuster estimate this. Their algorithm is shown in Figure 4.1. We've seen that Friedman and Schuster make three big design choices that are based on experimentation and heuristics:

- **Stopping Criteria** The `if` statement on Line 4 decides whether to continue branching or to stop, and in particular, the comparison of the signal to the noise is, in the words of the authors, "arbitrary". Also, the second pass done in `C4.5_Prune` (Line 17) may clip a branch and is based on the noisy sizes collected in Line 3.
- **Non-Leaf Queries** For nodes that are not leaves, the exponential mechanism is used with different utility functions. The question of

```
1  def dtree_private(D, atts, clss, d, e):
2    t = max over a in atts of len(a)
3    size = noisyCount(D, e/2)
4    if(d = 0 or len(atts) = 0 or size/(t*len(clss)) < sqrt(2)/e):
5      best_class = max over c in clss of
6        noisyCount(len(D[clss=c,]), e/2)
7      return Leaf(pred=best_class, size=size)
8    else:
9      U = map(attrs, lambda a: -c_entropy(a, clss, D))
10     best_att = exp_mech(domain=attrs, utilities=U, epsilon=e/2)
11     C = map(best_att, lambda a: dtree_private(D[best_att=a,],
12       att-best_att, clss, d-1, e))
13     return Node(att=best_att, children=C)
14
15 def dtree_pvt_top(D, atts, clss, d, budget):
16   t = dtree_private(D, atts, clss, d, budget/(d+1))
17   return C4.5_Prune(t)
```

Figure 4.1: Private C4.5 proposed by Friedman and Schuster [7].

which utility function to use arises, as this is unclear given their different sensitivities.

- **Number of Trees** It's well-known that a decision forest τ trees and a simple majority vote can often outperform a single decision tree. The number of trees to train in the DP setting could be much different than the answer in the non-private setting.

Another very important question is how much privacy budget to give to each of these components. If we give 0 budget to the exponential mechanism, then we are essentially picking a random attribute. Random decision forests have been known to outperform decision trees in the non-private setting. A summary of other works and their decisions on the above three criteria appear in the table below:

	Stopping Criteria in addition to $d = 0 \| k = 0$	ϵ_{stop}	Non-Leaf-Queries	ϵ_{NLQ}	ϵ_{LQ}	τ
Friedman & Schuster [7]	$\frac{D}{t*\|C\|} < \frac{\sqrt{2}}{\epsilon}$; d' user-defined; second pass with C4.5Prune	$\frac{\beta}{2d'}$	Exp. Mech with entropy, Gini, max	$\frac{\beta}{2d'}$	$\frac{\beta}{2d'}$	1
Mohammed et al. [17]	d' user-defined	0	Exp. Mech with entropy	$\frac{\beta}{d'}$	$\frac{\beta}{d'}$	1
Jagannathan et al. [10]	$d' = \min$ of $\frac{k}{2}$ and $\log_b(\|D\|) - 1$	0	Random	0	$\frac{\beta}{\tau}$	10
Patil & Singh [22]	$\frac{D}{t*\|C\|} < \frac{\sqrt{2}}{\epsilon}$; d' user-defined; second pass with C4.5Prune	$\frac{\beta}{2d'}$	Exp. Mech with entropy	$\frac{\beta}{2d'\tau}$	$\frac{\beta}{2d'\tau}$	10
Fletcher & Islam [5]	$\log_b(\|D\|) < \frac{\sqrt{2}\|C\|}{\epsilon}$	0	Random	0	$\frac{\beta}{\tau}$	m

Mohammed et al. [17] propose not collecting the size (Line 3) and eliminating the size criterion on Line 4. This allows us to use the entire ϵ budget for the node on the exponential mechanism and on the NoisyCount (Lines 10 and 6, respectively). Jagannathan et al. [10] propose choosing a random attribute on Line 9, spending no budget, and setting $d = \min\{\frac{k}{2}, \log_b(|D|) - 1\}$ where b is the average branching factor of the attributes, $\frac{1}{k}\sum_{i=1}^{k} |\mathcal{A}_i|$. Patil and Singh [22] do the same thing as Fletcher and Schuster but use multiple trees. Finally, Fletcher and Islam [5] do random trees but without a predefined depth; instead, they estimate the support at each node by assuming that datapoints are cut uniformly at each branch, and they stop when the support is less than the noise, similar to Friedman and Schuster.

4.0.2 Experiments

In this section, we plot the performances of the five algorithms on different datasets to highlight the existence of data dependence. Graphs of the data for different datasets appear in Figure 4.2. The datasets are described below:

Name	No. Points	No. Attrs	Avg. Branch Size	Class size
HIV-1 protease cleavage	1625	8	20	2
Nursery	12960	8	3.375	5
Contraceptive	1473	9	4.88	3

The HIV dataset is a sparse dataset; there are far more possible attribute values than the number of data points actually in the dataset. As we go farther down the decision tree, after only a short while, the vast majority of nodes will have no training data at all—after just 3 branches, there are 20^3 possible nodes which means almost 80% of the nodes have to be empty, and most of the nodes have large support [insert a graph of this]. As is well known in decision trees, low-support nodes give noisier estimates of the classes, and it rarely pays to branch at these nodes. This explains the performances of the algorithms: A1, having the most sophisticated stopping criteria, performs the best because it knows which nodes have low support. A2, having almost no stopping criteria at all, performs the worst (and takes the longest to train, going to all 20^d leaves), and A5, with a stopping criterion almost as weak, took too long to train.

The Nursery dataset, on the other hand, is dense. The stopping criteria does not matter as much as all nodes have a healthy support (in fact, the dataset has points for every possible value of the attributes). Thus, A2, which doesn't waste any budget on the stopping criterion, performs best.

The Contraceptive dataset is a mix between the two other datasets. Here, one advantage of A3 is very pronounced: its excellent performances at small values of ϵ. A3 gives the most budget to its leaf nodes, resulting in very accurate predictions at even small ϵ assuming the random trees are chosen in a fairly simple way. Because Contraceptive has no "bogus" attributes [insert graph of this], a randomly-chosen subset of the attributes is a fair set to work with, until the other, biased algorithms have enough budget to catch up.

In light of these experiments, we notice that stopping criteria, non-leaf-queries, and number of trees in the forest have a huge impact on the performance of the algorithm. What a programmer would really like to do is to shed light on these complicated relationships in an automated way. It would be easy for her to summarize all of the decision tree algorithms in a chart as in Figure 4.3. Each of the two places where she is uncertain is marked with a label, L1 or L2. The goal of `Jostle` is to learn automatically properties of the database that may lend to a certain decision at L1 or L2.

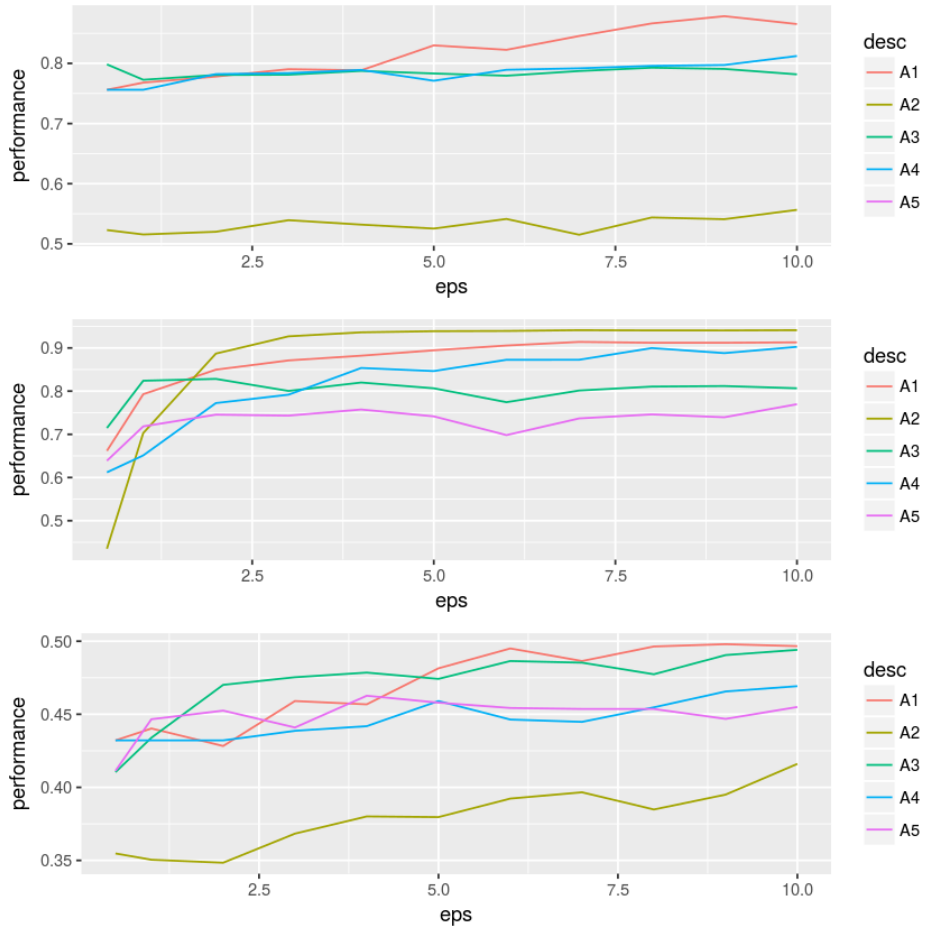

Figure 4.2: Performances of the five decision tree algorithms. The performance is measured from the prediction success rate on a validation set using a 30/70 validation to training split. Graph 1 does not have A5 because it takes such a long time to train, having a weak stopping criterion.

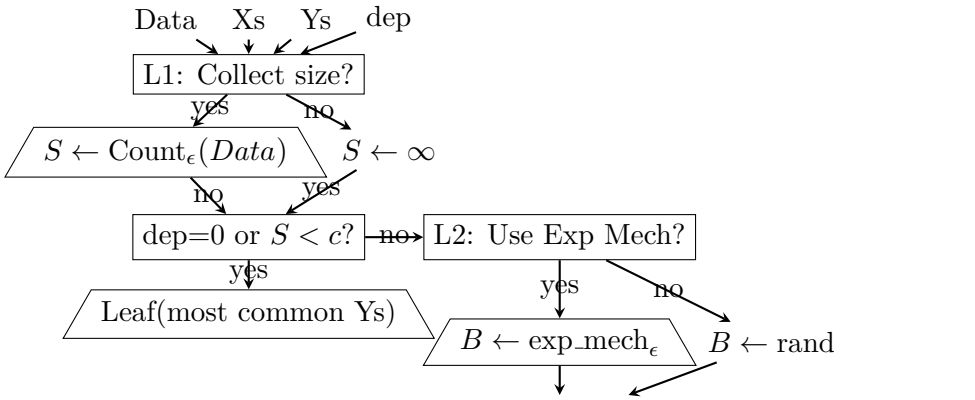

Figure 4.3: Our decision tree algorithm with NoisyConditionals which learn the execution path from past histories. Depending on what the NoisyConditionals say, this algorithm is capable of expressing all the decision tree algorithms in [6].

Our goal is to decide which of the heuristics provided by all the decision trees may perform best, along with any other heuristic the programmer may provide us.

Chapter 5

Future Work

We hope to extend our work to causal inference. We believe our approach is more general than Ashwin's work. We hope to implement synthesis.

Synthesis example:

```
1  class DAWA(Option):
2      def answer(db, queries):
3          #DAWA Implementation
4      def predict(F):
5          valuation = F.uniformity * (1-F.sparsity)
6          return valuation
7  class MWEM(Option):
8      def answer(db, queries):
9          #MWEM Implementation
10     def predict(F):
11         valuation = 1-(F.sparsity)^2
12         return valuation
```

However, our insights may not be perfect, and this is where `Jostle` can help. When public datasets are fed into `MkChoiceMaker`, traces on the algorithm are produced which the programmer can analyze and update features.synthesis on the metafeatures and `predict` methods is done to try to improve prediction accuracy. Perhaps `Jostle` realizes that a more accurate prediction for `MWEM` is $0.9 - (\texttt{sparsity})^{1.5}$. Or, perhaps it thinks a better definition for `sparsity` is the number of points more than one standard deviation higher than the mean. Or, perhaps it decides a metafeature `xyz` is better for predicting the accuracy of `MWEM`.

Bibliography

[1] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: The sulq framework. In *24th ACM SIGMOD International Conference on Management of Data / Principles of Database Systems, Baltimore (PODS 2005)*, Baltimore, Maryland, USA, June 2005.

[2] Kamalika Chaudhuri, Daniel Hsu, and Shuang Song. The large margin mechanism for differentially private maximization. 2, 09 2014.

[3] Kamalika Chaudhuri and Staal A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In *NIPS*, 2013.

[4] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[5] Sam Fletcher and Md Islam. A differentially private random decision forest using reliable signal-to-noise ratios, 11 2015.

[6] Sam Fletcher and Md Zahidul Islam. Decision tree classification with differential privacy: A survey. *CoRR*, abs/1611.01919, 2016.

[7] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 493–502, 2010.

[8] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on*

Management of Data, SIGMOD '16, pages 2101–2104, New York, NY, USA, 2016. ACM.

[9] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. *CoRR*, abs/1402.3329, 2014.

[10] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. A practical differentially private random decision tree classifier. In *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ICDMW '09, pages 114–121, Washington, DC, USA, 2009. IEEE Computer Society.

[11] Noah M. Johnson, Joseph P. Near, and Dawn Xiaodong Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[12] Fragkiskos Koufogiannis, Shuo Han, and George J. Pappas. Gradual release of sensitive data under differential privacy. *CoRR*, abs/1504.00429, 2015.

[13] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. A data- and workload-aware algorithm for range queries under differential privacy. *Proc. VLDB Endow.*, 7(5):341–352, January 2014.

[14] Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. Accuracy first: Selecting a differential privacy level for accuracy-constrained ERM. *CoRR*, abs/1705.10829, 2017.

[15] Xiaoqian Liu, Qianmu Li, Tao Li, and Dong Chen. Differentially private classification with decision tree ensemble. *Applied Soft Computing*, 62:807 – 816, 2018.

[16] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, September 2010.

[17] N. Mohammed, S. Barouti, D. Alhadidi, and R. Chen. Secure and private management of healthcare databases for data mining. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 191–196, June 2015.

[18] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[19] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, April 2014.

[20] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993.

[21] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. *SIGPLAN Not.*, 45(9):157–168, September 2010.

[22] Sanjay Singh and Abhijit Patil. Differential private random forest, 09 2014.

[23] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. *Proc. ACM Program. Lang.*, 1(ICFP):10:1–10:29, August 2017.