

Optimizing Hierarchical Queries for the Attribution Reporting API

Matthew Dawson, Badih Ghazi, Pritish Kamath, Kapil Kumar, Ravi Kumar, Bo Luan Pasin Manurangsi, Nishanth Mundru, Harikesh Nair, Adam Sealfon, Shengyu Zhu
Google
{mwdawson,prishk,kkapil,luanbo,pasin,nmundru,hsnair,adamsealfon,shengyuzhu}@google.com
{badihghazi,ravi.k53}@gmail.com

ABSTRACT

We study the task of performing hierarchical queries based on summary reports from the *Attribution Reporting API* for ad conversion measurement. We demonstrate that methods from optimization and differential privacy can help cope with the noise introduced by privacy guardrails in the API. In particular, we present algorithms for (i) denoising the API outputs and ensuring consistency across different levels of the tree, and (ii) optimizing the privacy budget across different levels of the tree. We provide an experimental evaluation of the proposed algorithms on public datasets.

KEYWORDS

ad conversion measurement, hierarchical aggregation, differential privacy, attribution reporting API

ACM Reference Format:

Matthew Dawson, Badih Ghazi, Pritish Kamath, Kapil Kumar, Ravi Kumar, Bo Luan, Pasin Manurangsi, Nishanth Mundru, Harikesh Nair, Adam Sealfon, Shengyu Zhu. 2023. Optimizing Hierarchical Queries for the Attribution Reporting API. In *Proceedings of The 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, AdKDD Workshop (AdKDD '23)*. ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

Over the last two decades, third-party cookies [25] have been essential to online advertising, and particularly to ad conversion measurement, whereby an ad impression (e.g., a click or a view) on a publisher site or app could be joined to a conversion on the advertiser, in order to compute aggregate conversion reports (e.g., the number of conversions attributed to a subset of impressions) or to train ad bidding models (e.g., [8, 16, 18, 22]). However, in recent years, privacy concerns have led several browsers to decide to deprecate third-party cookies, e.g., [23, 26, 27]. The *Attribution Reporting API* [1, 21] seeks to provide privacy-preserving ways for measuring ad conversions on the Chrome browser and the Android mobile operating system. This API relies on a variety of mechanisms for limiting the privacy leakage, including bounding the contributions to the output reports of the conversions attributed

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

AdKDD '23, August 06–10, 2023, Long Beach, CA
© 2023 Association for Computing Machinery.

to each impression, as well as noise injection to satisfy differential privacy (for more details, see Section 2).

We study the *conversion reporting* task, where a query consists of counting the number of conversions attributed to impressions such that some features of the conversion and the impression are restricted to certain given values. In particular, we focus on the *hierarchical queries* setting where the goal is to estimate the number of conversions attributed to impressions where the features are restricted according to certain nested conditions. Consider the example in Figure 1, where we wish to estimate the number of:

- Conversions attributed to impressions from campaign 123.
- Conversions that are also restricted to take place in New York.
- Conversions that are further restricted to occur on a Friday.

In general, the goal is to estimate the conversion count for each node in a given tree, similar to the one in Figure 1.

Such estimates can be obtained using summary reports from the Attribution Reporting API, as discussed in Section 2.2. In this work, we present a linear-time post-processing algorithm that denoises the estimates for different nodes that are returned by the API and ensures that the estimates are consistent with respect to the tree structure. We also show that our algorithm is optimal among all linear unbiased estimators for arbitrary trees, extending results for regular trees [9, 17] (Section 4). Since the API allows the ad-tech to allocate a privacy budget across different measurements containing contributions from the same impression, we provide an algorithm for optimizing the allocation of the privacy budget across the different levels of the tree (Section 5).

We start by recalling in Section 2 the basics of differential privacy and summarizing the query model for summary reports in the Attribution Reporting API. In Section 3, we formally define the optimization problem that we seek to solve. In Section 6, we provide an experimental evaluation of our algorithms on two public datasets. We conclude with some future directions in Section 7.

2 PRELIMINARIES

2.1 Differential Privacy (DP)

Let n be the number of rows in the dataset and let \mathcal{X} be the (arbitrary) set representing the domain of values for each row. We distinguish two types of columns (a.k.a. attributes): *known* and *unknown*. We also assume knowledge of the set of possible values that each unknown attribute can take.

Definition 2.1 (DP [11]). For $\epsilon \geq 0$, an algorithm \mathcal{A} is ϵ -DP if for every pair X, X' of datasets that differ on the unknown attributes of

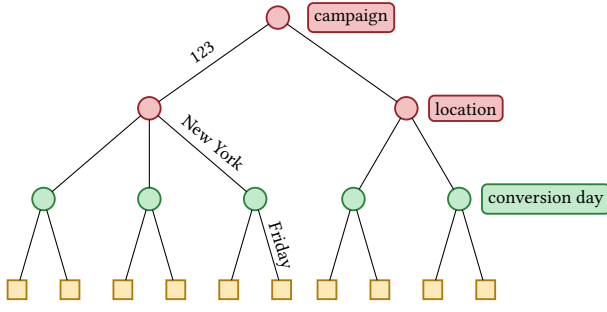


Figure 1: Example of Hierarchical Queries.

one row¹, and for every possible output o , it holds that $\Pr[\mathcal{A}(X) = o] \leq e^\epsilon \cdot \Pr[\mathcal{A}(X') = o]$.

LEMMA 2.2 (BASIC COMPOSITION). *Let \mathcal{A} be an algorithm that runs k algorithms $\mathcal{A}_1, \dots, \mathcal{A}_k$ on the same dataset such that \mathcal{A}_i is ϵ_i -DP with $\epsilon_i \geq 0$ for each $i \in [k]$. Then, \mathcal{A} is $(\sum_{i=1}^k \epsilon_i)$ -DP.*

LEMMA 2.3 (POST-PROCESSING). *Let $\epsilon > 0$, and R and R' be any two sets. If $\mathcal{A} : \mathcal{X}^n \rightarrow R$ is an ϵ -DP algorithm, and $f : R \rightarrow R'$ is any randomized mapping, then $(f \circ \mathcal{A}) : \mathcal{X}^n \rightarrow R'$ is ϵ -DP.*

For an extensive overview of DP, we refer the reader to the monograph [12]. A commonly used method in DP is the discrete Laplace mechanism. To define it, we recall the notion of ℓ_1 -sensitivity.

Definition 2.4 (ℓ_1 -sensitivity). Let \mathcal{X} be any set, and $f : \mathcal{X}^n \rightarrow \mathbb{R}^d$ be a d -dimensional function. Its ℓ_1 -sensitivity is defined as $\Delta_1 f := \max_{X, X'} \|f(X) - f(X')\|_1$, where X and X' are two datasets that differ on the unknown attributes of a single row.

Definition 2.5 (Discrete Laplace Mechanism). The *discrete Laplace distribution* centered at 0 and with parameter $a > 0$, denoted by $\text{DLap}(a)$, is the distribution whose probability mass function at integer k is $\frac{e^a - 1}{e^{a+1}} \cdot e^{-a|k|}$. The *d -dimensional discrete Laplace mechanism* with parameter a applied to a function $f : \mathcal{X}^n \rightarrow \mathbb{Z}^d$, on input a dataset $X \in \mathcal{X}^n$, returns $f(X) + Z$ where Z is a d -dimensional noise random variable whose coordinates are sampled i.i.d. from $\text{DLap}(a)$.

LEMMA 2.6. *For every $\epsilon > 0$, the d -dimensional discrete Laplace mechanism with parameter $a \leq \epsilon/\Delta_1 f$ is ϵ -DP.*

2.2 Attribution Reporting API

The *aggregatable reports* [1] are constructed as follows:

- **Impression (a.k.a. source) registration:** The API provides a mechanism for the ad-tech to register an impression (e.g., a click or view) on the publisher site or app. During registration, the ad-tech can specify impression-side aggregation keys (e.g., one corresponding to the campaign or geo location).
- **Conversion (a.k.a. trigger) registration:** The API also provides a mechanism for the ad-tech to register a conversion on

¹We note that this instantiation of the DP definition is related to the label DP setting in machine learning [7, 13–15], where the features of an example are considered known and only its label is deemed unknown and sensitive. In our use case, there might be multiple unknown attributes, whose concatenation is treated in a conceptually similar way to the *label* in the label DP setting.

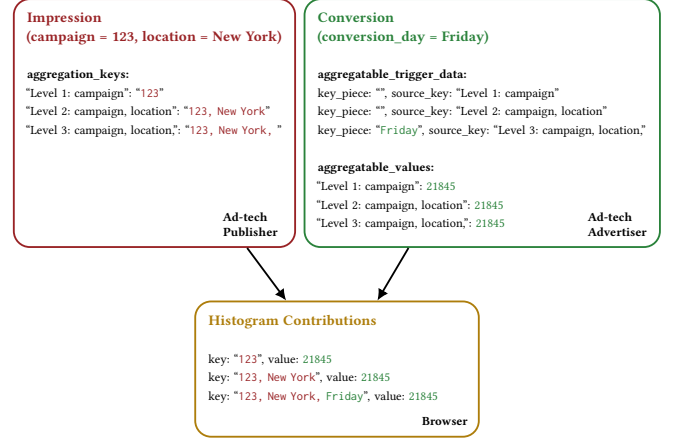


Figure 2: Histogram Contributions Generation.

the advertiser site or app. As it does so, the ad-tech can specify conversion-side key pieces along with aggregatable values corresponding to each setting of the impression-side aggregation keys. E.g., a conversion-side key piece could capture the conversion type or (a discretization of) the conversion timestamp. The combined aggregation key (which can be thought of as the concatenation of the impression-side aggregation key and the conversion-side key piece) is restricted to be at most 128 bits. The aggregatable value is required to be an integer between 1 and the L_1 parameter of the API, which is set to $2^{16} = 65,536$.

- **Attribution:** The API supports last-touch attribution where the conversion is attributed to the last (unexpired) impression registered by the same ad-tech. (The API supports a broader family of single-touch attribution schemes allowing more flexible prioritization over impressions and conversions; we do not discuss this aspect any further as it is orthogonal to our algorithms.²)
- **Histogram contributions generation:** The API enforces that the sum of contributions across all aggregatable reports generated by different conversions attributed to the same impression is capped to at most the $L_1 = 2^{16}$ parameter. An example construction of histogram contributions is in Figure 2. While in this example the conversion key piece depends only on conversion information (the conversion day), it can be replaced by an attribute that depends on both impression and conversion information (e.g., a discretization of the difference between impression time and conversion time); this can be done using filters [4].

At query time, a set of histogram keys is requested by the ad-tech [6]. (The set of keys that are queried could be set to the Cartesian product of all *known* values of the impression-side features with the set of all *possible* values of the conversion-side features.) The aggregatable reports are combined in the aggregation service to produce a *summary report* by applying the discrete Laplace mechanism (Definition 2.5) with parameter ϵ/L_1 to the requested aggregation keys. This report satisfies ϵ -DP where each row of the dataset corresponds to an impression and its attributed conversions if any

²The API moreover enforces a set of rate limits on registered impressions, and attributed conversions; we omit discussing these since they are not essential to the focus of this paper. We refer the interested reader to the documentation [2].

(see Table 1 for an example³), and where the known columns in Definition 2.1 are the attributes that only depend on impression information (campaign and location in Table 1).

Click	campaign	location	conversion day
1	123	Paris	Monday
2	456	Chicago	Friday
3	789	London	*
4	123	New York	Friday
...

Table 1: Dataset Post-Attribution and Pre-Aggregation.

In the hierarchical aggregation setting, each node of the tree corresponds to an aggregation key, and the level of the node is (implicitly) specified in the impression-side aggregation key and/or in the conversion-side key piece (as shown in Figure 2).

For the use case of estimating conversion counts, the aggregatable value could be set so as to increment the count by +1. Since the scale of the noise injected in summary reports is L_1/ϵ , the ad-tech can improve accuracy by setting the contribution of an increment to $+L_1$ instead of +1 (and then scaling down the value it receives from the aggregation service by L_1). If the L_1 contribution has to be divided across multiple keys, the contribution of each increment needs to be scaled down accordingly. E.g., in Figure 2, since each impression affects 3 keys, the contribution is set to $\lfloor L_1/3 \rfloor = 21,845$.

3 OPTIMIZATION PROBLEM

3.1 Hierarchical Query Estimation

We formally define the *hierarchical query estimation* problem. Given a dataset X , consider a tree where each node corresponds to the subset of the rows of X , conditioned on the values of some of the attributes. We consider the setting where each level of the tree introduces a conditioning on the value of a new attribute. For *known* attributes, the child nodes of a node correspond to the different values taken by that attribute in X within the rows. For *unknown* attributes, the child nodes correspond to *all* possible values for that attribute, whether or not they actually occur in the dataset. Given this tree, the problem is to privately release the approximate number of data rows corresponding to each node that have an attributed conversion.

3.2 Error Measure and Consistency

We consider the following error measure, which is defined in [20].

Definition 3.1 (RMS Relative Error at Threshold). For a count $c \geq 0$, and its randomized estimate $\hat{c} \in \mathbb{R}$, the *Root Mean Squared Relative Error at Threshold τ* when estimating c by \hat{c} is defined as

$$\text{RMSRE}_\tau(c, \hat{c}) := \sqrt{\mathbb{E} \left[\left(\frac{|\hat{c} - c|}{\max(\tau, c)} \right)^2 \right]},$$

where the expectation is over the randomness of \hat{c} .

³The * in the conversion-related field in Table 1 indicates that the click corresponding to that row did not get an attributed conversion.

Suppose we have the count estimates (e.g., the number of conversions as in Figure 1) at every node in a tree. Each conversion contributes to the count for multiple nodes. For example, a conversion for ad campaign 123 that occurs on a Friday in New York contributes to the 6th leaf from the left, but also to each of its ancestor nodes. This imposes relationships among the counts at various nodes. If the only geo locations with conversions attributed to ad campaign 123 on Friday are New York and Chicago, then the total number of such conversions must equal the sum of the number of such conversions in each of the two locations. More generally, the count for any node must equal the sum of the counts for its children. An estimator with this property is called *consistent*.

For a tree T with levels L_0, \dots, L_d , with L_i being the set of nodes at level i , and estimators \hat{c}_v of the counts c_v at each node v , define the tree error to be

$$\text{RMSRE}_\tau(T) := \sqrt{\frac{1}{d+1} \sum_{i=0}^d \left(\frac{1}{|L_i|} \sum_{v \in L_i} \text{RMSRE}_\tau(c_v, \hat{c}_v)^2 \right)}.$$

The goal of the hierarchical query estimation problem is to privately estimate the counts of every node with minimum possible tree error, where the estimates should be consistent. To achieve this, we will employ post-processing and privacy budgeting strategies.

4 POST-PROCESSING ALGORITHMS

Directly applying the discrete Laplace mechanism to add independent noise to each node does not result in a consistent estimate. Consistency can be achieved by estimating only the counts of leaves and inferring the count of each nonleaf by adding the counts of its leaf descendants, but this can lead to large error for nodes higher up in the tree. Alternatively, one can achieve consistency by post-processing independent per-node estimates. Since DP is preserved under post-processing (Lemma 2.3), this comes at no cost to the privacy guarantee, and it can substantially improve accuracy.

Since the count of any node must equal the sum of the counts of its children, we can obtain a second independent estimate of the count of any nonleaf node by summing the estimates of its children. We can combine these two estimates to obtain a single estimate of lower variance. Extending this observation, Hay et al. [17] and Cormode et al. [9] give efficient post-processing algorithms for regular (every non-leaf has the same number of children) and balanced (every leaf is at the same depth) trees, achieving consistency and also a substantial improvement in accuracy. In particular, estimating the counts of each node can be expressed as a linear regression problem, and these algorithms compute the least-squares solution, which is known to achieve optimal error variance among all unbiased linear estimators. Moreover, this special case of least-squares regression can be solved in linear time.

We generalize this algorithm to arbitrary trees. This allows us to handle trees with different fanouts at different levels or even at different nodes in the same level, which is the case for the conversion reporting trees we study. See Algorithm 1.

The algorithm comprises two linear-time passes. The first pass proceeds bottom-up from the leaves to the root. For each node it computes the optimal convex combination of the direct estimate at that node and the recursively computed estimates at each of its children, where the weights of the convex combination are

Algorithm 1 Post-processing for denoising & consistency

Params: Tree T , variance var_v for estimates of node v .
Input: Estimated counts x_v for each node v in the tree.
Output: De-noised estimates y_v for each node v in the tree.

```

for  $v \in T$ : let  $z_v \leftarrow x_v$  and  $\widehat{var}_v \leftarrow var_v$ .
# Bottom-up pass through nonleaf nodes
for each internal node  $v$  from largest to smallest depth do
  Let  $ch\_var_v \leftarrow \sum_{u \in \text{child}(v)} var_u$  and  $ch\_z_v \leftarrow \sum_{u \in \text{child}(v)} z_u$ 
  Let  $z_v \leftarrow \frac{ch\_var_v}{var_v + ch\_var_v} \cdot z_v + \frac{var_v}{var_v + ch\_var_v} \cdot ch\_z_v$ .
  Let  $\widehat{var}_v \leftarrow \frac{var_v \cdot ch\_var_v}{var_v + ch\_var_v}$ .
# Top-down pass through nonroot nodes
Let  $y_r \leftarrow z_r$ , where  $r$  is the root.
for each non-root node  $v$  from smallest to largest depth do
  Let  $p \leftarrow \text{parent}(v)$ 
  Let  $y_v \leftarrow z_v + \frac{\widehat{var}_v}{ch\_var_p} \cdot (y_p - ch\_z_p)$ 
return  $(y_v)_{v \in T}$ 

```

inversely proportional to the variances. This improves the error of the estimate at each node. The second pass proceeds top-down from the root to the leaves and can be thought of as a projection step. For each node it computes the difference between the estimate at that node and the sum of the estimates of its children, and divides this difference equally among the children. This directly achieves consistency, as well as further reduces the error of the estimates.

This procedure not only reduces the variance of each estimate but also achieves the best possible among all unbiased linear estimators.

THEOREM 4.1 (OPTIMALITY OF POST-PROCESSING). *For every $v \in T$, y_v is the best linear unbiased estimator (BLUE) of the count c_v . In particular, among all linear unbiased estimators, $(y_v)_{v \in T}$ minimizes $\text{RMSRE}_\tau((c_v)_{v \in T}, (y_v)_{v \in T})$.*

This extends the results from [9, 17], which work only for regular trees, to arbitrary trees. Similar to their proofs, the theorem above follows once we show that the (appropriately scaled) estimates $(y_v)_{v \in T}$ are an ordinary least squares estimator (OLS) of the counts $(c_v)_{v \in T}$. Proving the latter boils down to showing that the estimates satisfy the two conditions in the following lemma:

LEMMA 4.2. *For any T , $(var_v)_{v \in T}$ and $(x_v)_{v \in T}$, let y_v be the output of Algorithm 1. Then, the following hold:*

- (Consistency) For every internal node v , $y_v = \sum_{u \in \text{child}(v)} y_u$.
- (Weighted Root-to-Leaf Sum Preservation) For every leaf v , $\sum_{u \in \text{anc}(v)} \frac{x_u}{var_u} = \sum_{u \in \text{anc}(v)} \frac{y_u}{var_u}$, where $\text{anc}(v)$ denotes the nodes on the path from v to r (inclusive).

The proof of Lemma 4.2 is by induction on the number of nodes in the tree. The inductive step is done by selecting a node whose children are all leaves and “coalescing” all of the node’s children. We omit the full proof from this version due to space constraints.

5 PRIVACY BUDGETING OVER TREE LEVELS

To allocate the privacy budget across the levels of the tree, the simplest approach is to divide it equally among the levels, or to put all of the budget on the lowest level. However, basic composition

(Lemma 2.2) allows us to allocate the privacy budget arbitrarily among the nodes of the tree, and we can apply post-processing (Algorithm 1) to noisy initial estimates with unequal variances as well. This motivates the question of whether we can improve accuracy with an unequal privacy budget allocation, and if so, how.

Given the true counts c , we give a greedy iterative approach for optimizing the privacy budget allocation. Let k be a number of phases (e.g., $k = 20$) corresponding to the granularity of the allocation. Initially allocate zero (or infinitesimal) privacy budget to each level, and divide the (remaining) privacy budget into k units of size ϵ/k . In each of k phases, select the level that would result in the lowest $\text{RMSRE}_\tau(T)$ when using ϵ/k additional privacy budget, and increase the privacy budget of that level by ϵ/k . The error $\text{RMSRE}_\tau(T)$ can be estimated empirically, or computed directly using the variance of each post-processed estimate \hat{c}_v , which can be computed using an extension to Algorithm 1; due to space constraints, we defer the details to the full version.

Using the true counts to optimize the privacy budget allocation can leak sensitive information and violate the privacy guarantee, and is infeasible in the API. Instead of using the true counts c to allocate the privacy budget, one can use alternatives such as simulated data, or historical data that is not subject to the privacy constraints (e.g., before third-party cookie deprecation), or noisy historical data that has already been protected with DP (e.g., the output of the API over data from a previous time period). We refer to this family of privacy budget optimization methods as *prior-based*. When no such alternatives are available, one could start out with a suboptimal privacy budgeting strategy (e.g., uniform privacy budgeting across levels) and improve the allocation over time.

6 EXPERIMENTAL EVALUATION

We evaluate the algorithms on two public ad conversion datasets.

Criteo Sponsored Search Conversion Log (CSSCL) Dataset [24]. This dataset consists of 15,995,634 points obtained from a sample of 90-day logs of live traffic from Criteo Predictive Search. Each point contains information on a user action (e.g., a click on an ad) and a potential subsequent conversion (purchase of the corresponding product) within a 30-day attribution window. We consider the following attributes: `partner_id`, `product_country`, `device_type`, `product_age_group`, and `Time_delay_for_conversion`. The last attribute is discretized into 2-day (or 6-day) buckets so that there are at most 15 (5 respectively) possible values of the rounded time delay. The first 4 attributes are considered *known* (they only relate to the impression), whereas the last is considered *unknown*. For the discretization into 6-day buckets, we retain the unknown attribute and 3 known attributes instead of 4 (omitting `product_age_group`), resulting in a depth 4 tree (instead of depth 5).

In Figure 3, we plot RMSRE_τ versus ϵ for various different methods evaluated on the later 45 days of data, namely,

- Equal privacy budget split across levels with no post-processing.
- Equal privacy budget split across levels with post-processing.
- All privacy budget on leaves with post-processing.
- Prior-based optimized privacy budget split across levels optimized for each `partner_id` with no post-processing.
- Prior-based optimized privacy budget split across levels optimized for each `partner_id` with post-processing.

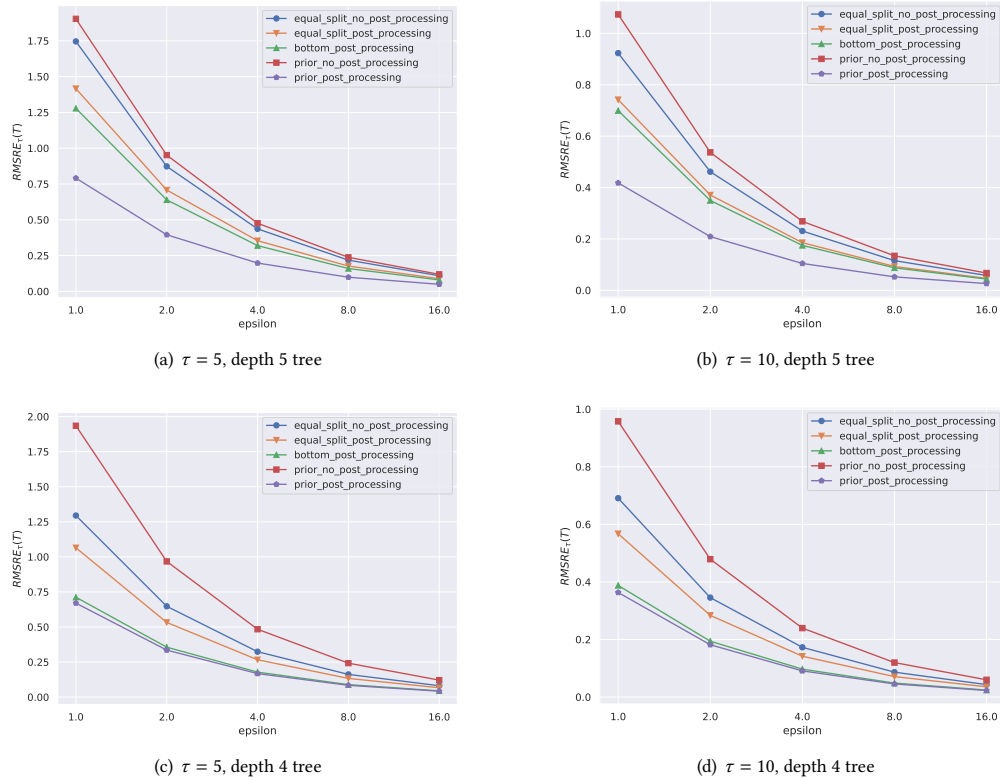


Figure 3: $RMSRE_\tau(T)$ vs ϵ for $\tau \in \{5, 10\}$ on CSSCL dataset, with noisy prior obtained via equal budget split and $\epsilon = 1$.

For the prior-based privacy budget split optimization, the privacy budgeting was performed using a noisy prior computed on the first 45 days (privately estimated with $\epsilon = 1$ and an equal privacy budget split over levels). For `partner_id`'s that appear only in the later 45 days, the prior is computed on all the `partner_id`'s that appear in the first 45 days of data. E.g., for $\epsilon = 4$ and the depth 5 tree, $RMSRE_{10} \approx 0.1$, and for the depth 4 tree, $RMSRE_5 \approx 0.17$.

Criteo Attribution Modeling for Bidding (CAMB) Dataset [19]. It consists of $\sim 16M$ impressions from 30 days of Criteo live traffic. We consider last-touch attribution, and impression attributes: `campaign`, categorical features `cat1`, `cat8`, and a discretization of conversion delay, i.e., the difference between `conversion_timestamp` and `(impression) timestamp`. As for CSSCL, we consider two discretizations for the difference, with two tree depths. Figure 4 shows the plots for CAMB. The privacy budgeting was performed similarly to CSSCL but with the noisy prior computed on the first 15 days. For $\epsilon = 4$ and the depth 4 tree, $RMSRE_{10} \approx 0.19$, and for the depth 3 tree (omitting attribute `cat8`), $RMSRE_5 \approx 0.20$ and $RMSRE_{10} \approx 0.12$.

Our prior-based budgeting with post-processing method equals or outperforms all other approaches in each setting (Figures 3 and 4).

7 CONCLUSION AND FUTURE DIRECTIONS

In this work, we studied hierarchical querying of the Attribution Reporting API, and presented algorithms for consistency enforcement and privacy budgeting, demonstrating their performance on

two public ad datasets. We next discuss some interesting future directions. We focused on the so-called OPC (“one per click”) setting where each impression gets at most a single attributed conversion. An important direction is to consider the extension to the more general MPC (“many per click”) setting where an impression can get multiple attributed conversions. It would also be interesting to extend our treatment of conversion counts to the task of estimating conversion values. While the error is small for values of ϵ around 16 in our evaluation, we note that this is specific to the datasets and the (restricted) functionality that we study (i.e., conversion counts with OPC). Achieving small errors on additional functionalities (e.g., MPC or conversion values) will likely require larger values of ϵ . Another natural direction is to extend the consistency enforcement algorithm to ensure monotonicity (i.e., that the output estimate for a node of the tree is at least the output estimate for any of its children), and non-negativity. Our privacy budgeting method optimizes for one privacy parameter *per-level* of the tree; it would be good to explore the extent to which *per-node* privacy budgeting can yield higher accuracy. While we considered *data-independent* weights when defining the tree error in terms of the node errors, there could be situations where *data-dependent* weights are preferable, e.g., to avoid the tree error being dominated by the error in many nodes with no conversions, while being insensitive to the error of few nodes where most of the conversions occur. Another

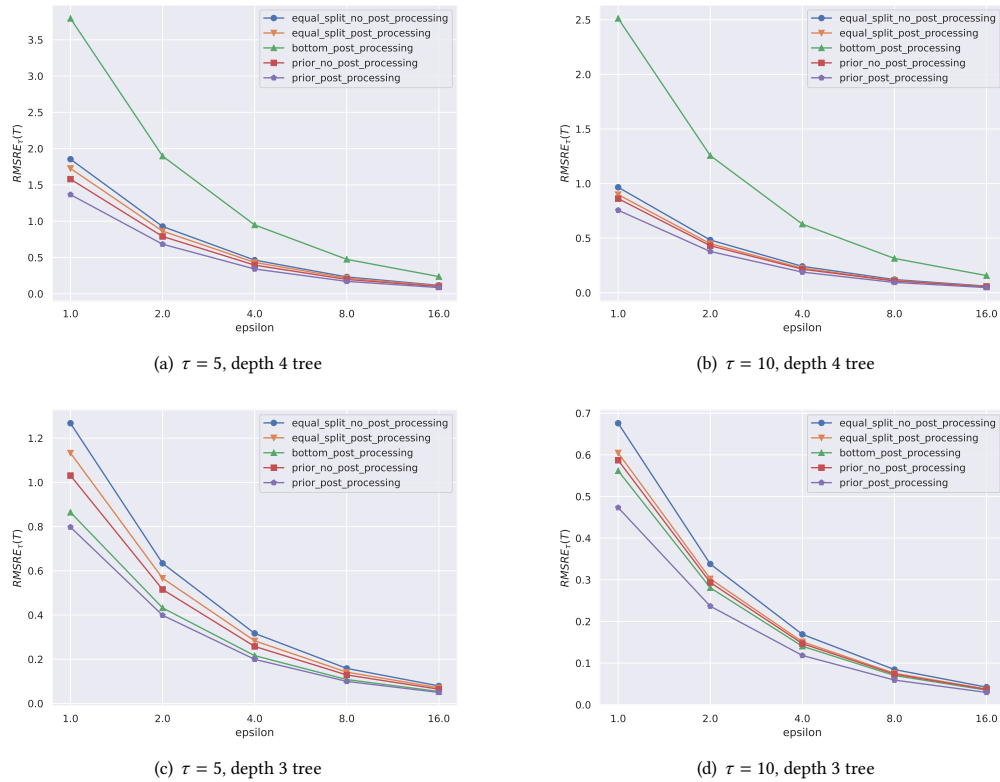


Figure 4: $\text{RMSRE}_\tau(T)$ vs ϵ for $\tau \in \{5, 10\}$ on CAMB dataset with noisy prior obtained via equal budget split and $\epsilon = 1$.

interesting direction to study privacy budgeting under approximate DP [10]. Our work considered the hierarchical query model; a natural direction is to optimize the *direct query model* [5]. Finally, the Attribution Reporting API also offers *event-level reports* [3]. It would be interesting to see if these could be also used to further improve the accuracy for hierarchical queries.

REFERENCES

- [1] Attribution reporting: Aggregatable reports api. <https://developer.android.com/design-for-safety/privacy-sandbox/attribution#aggregatable-reports-api>.
- [2] Attribution Reporting Draft Community Group Report. <https://wicg.github.io/attribution-reporting-api/>.
- [3] Attribution Reporting: Event-level Reports. <https://developer.android.com/design-for-safety/privacy-sandbox/attribution#event-level-reports>.
- [4] Attribution reporting: Trigger filters. <https://developer.android.com/design-for-safety/privacy-sandbox/attribution#trigger-filters>.
- [5] Privacy Sandbox Aggregation Service: Query Models. <https://github.com/google/privacy-sandbox-aggregation-service#query-models>.
- [6] Set up Aggregation Service for Aggregatable Reports: Collect and Batch Aggregatable Reports. <https://github.com/privacysandbox/aggregation-service#collect-and-batch-aggregatable-reports>.
- [7] CHAUDHURI, K., MONTELEONI, C., AND SARWATE, A. D. Differentially private empirical risk minimization. *JMLR* 12, 3 (2011).
- [8] CHOI, Y., KWON, M., PARK, Y., OH, J., AND KIM, S. Delayed feedback model with negative binomial regression for multiple conversions. In *ADKDD* (2020).
- [9] CORMODE, G., PROCOPIUC, C., SRIVASTAVA, D., SHEN, E., AND YU, T. Differentially private spatial decompositions. In *ICDE* (2012), pp. 20–31.
- [10] DWORK, C., KENTHAPADI, K., MCSHERRY, F., MIRONOV, I., AND NAO, M. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT* (2006).
- [11] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. D. Calibrating noise to sensitivity in private data analysis. In *TCC* (2006), pp. 265–284.
- [12] DWORK, C., ROTH, A., ET AL. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
- [13] ESFANDIARI, H., MIRROKNI, V., SYED, U., AND VASSILVITSKII, S. Label differential privacy via clustering. In *AISTATS* (2022), pp. 7055–7075.
- [14] GHAZI, B., GOLOWICH, N., KUMAR, R., MANURANGSI, P., AND ZHANG, C. Deep learning with label differential privacy. *NeurIPS* (2021), 27131–27145.
- [15] GHAZI, B., KAMATH, P., KUMAR, R., LEE, E., MANURANGSI, P., VARADARAJAN, A., AND ZHANG, C. Regression with label differential privacy. In *ICLR* (2023).
- [16] GU, T., KUANG, K., ZHU, H., LI, J., DONG, Z., HU, W., LI, Z., HE, X., AND LIU, Y. Estimating true post-click conversion via group-stratified counterfactual inference, 2021.
- [17] HAY, M., RASTOGI, V., MIKLAU, G., AND SUCIU, D. Boosting the accuracy of differentially-private histograms through consistency. *VLDB* (2010).
- [18] LU, Q., PAN, S., WANG, L., PAN, J., WAN, F., AND YANG, H. A practical framework of conversion rate prediction for online display advertising. In *ADKDD*. 2017.
- [19] MEYNET, J. Criteo attribution modeling for bidding dataset. <https://ailab.criteo.com/criteo-attribution-modeling-bidding-dataset/>.
- [20] NADAN, A., WHITE, A., CUCU, A., NALPAS, M., AND MASTROMATTO, Z. Experiment with summary report design decisions, November 2022. <https://developer.chrome.com/docs/privacy-sandbox/summary-reports/design-decisions/>.
- [21] NALPAS, M., AND WHITE, A. Attribution Reporting, May 2021. <https://developer.chrome.com/en/docs/privacy-sandbox/attribution-reporting/>.
- [22] QIU, Y., TZIORTZIOTIS, N., HUE, M., AND VAZIRGIANNIS, M. Predicting conversions in display advertising based on URL embeddings. *arXiv preprint:2008.12003* (2020).
- [23] SCHUH, J. Building a more private web: A path towards making third party cookies obsolete, January 2020. <https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html>.
- [24] TALLIS, M., AND YADAV, P. Reacting to variations in product demand: An application for conversion rate (CR) prediction in sponsored search. In *IEEE BigData* (2018).
- [25] WIKIPEDIA. HTTP cookie: third-party cookie. https://en.wikipedia.org/wiki/HTTP_cookie#Third-party_cookie. [Online; accessed 25-May-2023].
- [26] WILANDER, J. Full Third-Party Cookie Blocking and More, March 2020. <https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/>.
- [27] WOOD, M. Today's Firefox Blocks Third-Party Tracking Cookies and Cryptomining by Default, 2019. <https://blog.mozilla.org/en/products/firefox/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/>.