Optimizing Hierarchical Queries for the Attribution Reporting API

Matthew Dawson[1], Badih Ghazi[1,*], Pritish Kamath[1], Kapil Kumar[1], Ravi Kumar[1], Bo Luan[1], Pasin Manurangsi[1], Nishanth Mundru[1], Harikesh Nair[1], Adam Sealfon[1] and Shengyu Zhu[1]

[1]*Google*

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

1. Introduction

Over the last two decades, third-party cookies [1] 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., [2, 3, 4, 5]). However, in recent years, privacy concerns have led several browsers to decide to deprecate third-party cookies, e.g., [6, 7, 8]. The *Attribution Reporting API* [9, 10] 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 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.
- that are also restricted to take place in New York.
- 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.

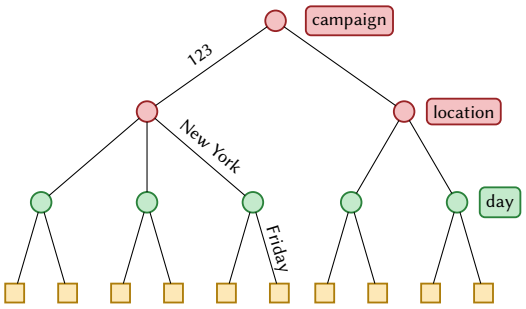


Figure 1: Example of Hierarchical Queries.

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 [11, 12] (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

AdKDD'23

mwdawson@google.com (M. Dawson); badihghazi@gmail.com (B. Ghazi); pritishk@google.com (P. Kamath); kkapil@google.com (K. Kumar); ravi.k53@gmail.com (R. Kumar); luanbo@google.com (B. Luan); pasin@google.com (P. Manurangsi); nmundru@google.com (N. Mundru); hsnair@google.com (H. Nair); adamsealfon@google.com (A. Sealfon); shengyuzhu@google.com (S. Zhu)

CEUR Workshop Proceedings (CEUR-WS.org)

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 1 (DP [13]). *For $\varepsilon \geq 0$, an algorithm $\mathcal{A}$ is ε-DP if for every pair X, X' of datasets that differ on the unknown attributes of one row[1], and for every possible output o, it holds that $\Pr[\mathcal{A}(X) = o] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(X') = o]$.*

Lemma 2 (Basic Composition). *Let $\mathcal{A}$ be an algorithm that runs k algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_k$ on the same dataset such that $\mathcal{A}_i$ is ε_i-DP with $\varepsilon_i \geq 0$ for each $i \in [k]$. Then, $\mathcal{A}$ is $(\sum_{i=1}^{k} \varepsilon_i)$-DP.*

Lemma 3 (Post-processing). *Let $\varepsilon > 0$, and R and R' be any two sets. If $\mathcal{A} : \mathcal{X}^n \to R$ is an ε-DP algorithm, and $f : R \to R'$ is any randomized mapping, then $(f \circ \mathcal{A}) : \mathcal{X}^n \to R'$ is ε-DP.*

For an extensive overview of DP, we refer the reader to the monograph [18]. A commonly used method in DP is the discrete Laplace mechanism. To define it, we recall the notion of ℓ_1-sensitivity.

Definition 4 (ℓ_1-sensitivity). *Let $\mathcal{X}$ be any set, and $f : \mathcal{X}^n \to \mathbb{R}^d$ 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 5 (Discrete Laplace Mechanism). *The* discrete Laplace distribution *centered at 0 and with parameter $a > 0$, denoted by $\mathrm{DLap}(a)$, 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 $f : \mathcal{X}^n \to \mathbb{Z}^d$, 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 $\mathrm{DLap}(a)$.*

Lemma 6. *For every $\varepsilon > 0$, the d-dimensional discrete Laplace mechanism with parameter $a \leq \varepsilon / \Delta_1 f$ is ε-DP.*

2.2. Attribution Reporting API

The *aggregatable reports* [10] 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 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

[1] We note that this instantiation of the DP definition is related to the label DP setting in machine learning [14, 15, 16, 17], 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.

[2] The 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 [19].

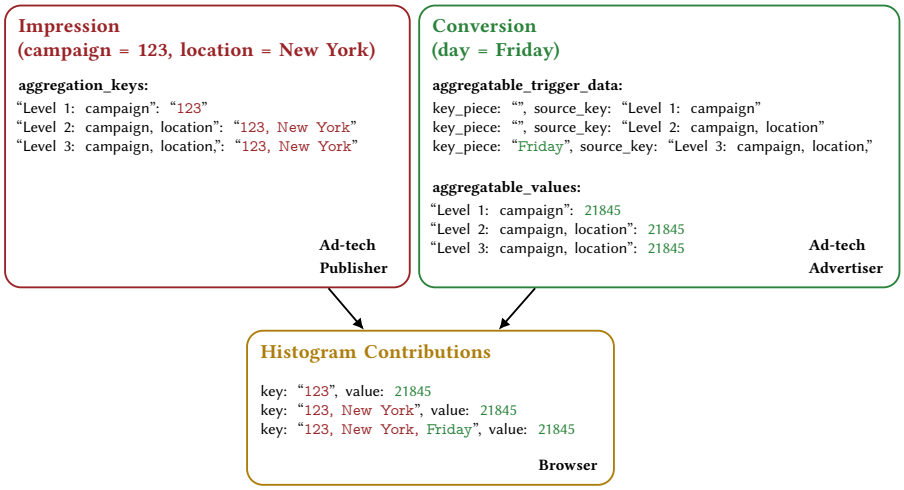


Figure 2: Histogram Contributions Generation.

Table 1
Dataset Post-Attribution and Pre-Aggregation.

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

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 [20].

At query time, a set of histogram keys is requested by the ad-tech [21]. (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 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 (see Table 1 for an example[3]), and where the known columns in Definition 1 are the attributes that only depend on impression information (campaign and location in Table 1).

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

3.2. Error Measure and Consistency

We consider the following error measure, which is defined in [22].

Definition 7 (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* $\mathrm{RMSRE}_\tau(c, \hat{c}) := \sqrt{\mathbb{E}\left[\left(\frac{|\hat{c}-c|}{\max(\tau,c)}\right)^2\right]}$*, where the expectation is over the randomness of $\hat{c}$.*

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 $L_0, \ldots, L_d$, 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 $\mathrm{RMSRE}_\tau(T)$ to be

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

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 node 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 3), this comes at no cost to the privacy guarantee, and it can substantially improve accuracy.

Algorithm 1 CombineEstimates

Input: $(x; \mathsf{var}_x)$, $(y; \mathsf{var}_y)$: x, y are samples from random variables X, Y resp. such that $\mathbb{E}\,X = \mathbb{E}\,Y$ and variances $\mathsf{Var}(X) = \mathsf{var}_x$ and $\mathsf{Var}(Y) = \mathsf{var}_y$.
Output: $(z; \mathsf{var}_z)$: combined estimate and variance.

$\mathsf{var}_z \leftarrow \frac{\mathsf{var}_x \cdot \mathsf{var}_y}{\mathsf{var}_x + \mathsf{var}_y}$
$z \leftarrow \mathsf{var}_z \cdot \left(\frac{x}{\mathsf{var}_x} + \frac{y}{\mathsf{var}_y}\right)$
return $(z; \mathsf{var}_z)$

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. [11] and Cormode et al. [12] 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 in Algorithm 2. This allows us to handle trees with different fanouts and noise at different levels or even at different nodes in the same level, which is the case for the conversion reporting trees we study. The input to the algorithm are independent noisy counts x_v for all nodes v in T, with $\mathbb{E}\,x_v = c_v$ and variance $\mathsf{Var}(x_v) = \mathsf{var}_v$.

The key idea is a simple method (Algorithm 1) to combine two *independent* and *unbiased* estimates of the same quantity to get an estimate with reduced variance as follows: Suppose X and Y are two *independent* random variables with $\mathbb{E}\,X = \mathbb{E}\,Y = C$, with variances var_X and var_Y respectively, then the optimal convex combination of X and Y that minimizes the variance in the estimate of C is inversely proportional to the variances, namely $Z = (\mathsf{var}_X \cdot Y + \mathsf{var}_Y \cdot X)/(\mathsf{var}_X + \mathsf{var}_Y)$, which has variance $\mathsf{Var}(Z) = (\mathsf{var}_X \cdot \mathsf{var}_Y)/(\mathsf{var}_X + \mathsf{var}_Y)$.

Algorithm 2 comprises of two linear-time passes. The first pass proceeds bottom-up from the leaves to the root. For each non-leaf node v it recursively computes $z_v^{\uparrow}$, which is an optimal linear unbiased estimate of c_v, using only the noisy counts x_u for all u that are in the sub-tree rooted at v, *excluding* v. This is combined with the noisy count x_v to compute $z_v^{\Uparrow}$, which is an optimal linear unbiased estimate of c_v, using only the noisy counts x_u for all u that are in the sub-tree rooted at v, *including* v.

Algorithm 2 TreePostProcessing

Params: Tree T with root r.
Input: $(x_v; \mathsf{var}_v)_{v \in T}$: noisy counts and variances for all $v \in T$.
Output: $(\widehat{x}_v; \mathsf{var}_v)_{v \in T}$: post-processed estimates and variance for all v.

 # Bottom-up pass
for leaf $v \in T$ **do**
  $\triangleright\ (z_v^{\Uparrow}; \mathsf{var}_v^{\Uparrow}) \leftarrow (x_v; \mathsf{var}_v)$
for each internal node v from largest to smallest depth **do**
  $\triangleright\ (z_v^{\uparrow}; \mathsf{var}_v^{\uparrow}) \leftarrow \left(\sum_{u \in \mathrm{child}(v)} z_u^{\Uparrow}; \sum_{u \in \mathrm{child}(v)} \mathsf{var}_u^{\Uparrow}\right)$
  $\triangleright\ (z_v^{\Uparrow}; \mathsf{var}_v^{\Uparrow}) \leftarrow \mathsf{CombineEstimates}((x_v; \mathsf{var}_v), (z_v^{\uparrow}; \mathsf{var}_v^{\uparrow}))$

 # Top-down pass
For root r:
$\triangleright\ (\widehat{x}_r; \widehat{\mathsf{var}}_r) \leftarrow (z_r^{\Uparrow}; \mathsf{var}_r^{\Uparrow})$
$\triangleright\ (z_r^{\Downarrow}; \mathsf{var}_r^{\Downarrow}) \leftarrow (x_r; \mathsf{var}_r)$
for each non-root node v from smallest to largest depth **do**
  $\triangleright\ p \leftarrow \mathrm{parent}(v)$
  $\triangleright\ (z_v^{\downarrow}; \mathsf{var}_v^{\downarrow}) \leftarrow \left(z_p^{\Downarrow} - \sum_{u \in \mathrm{child}(p) \setminus \{v\}} z_u^{\Uparrow}; \mathsf{var}_p^{\Downarrow} + \sum_{u \in \mathrm{child}(p) \setminus \{v\}} \mathsf{var}_u^{\Uparrow}\right)$ *# equals* $(z_p^{\Downarrow} - z_p^{\uparrow} + z_v^{\Uparrow}; \mathsf{var}_p^{\Downarrow} + \mathsf{var}_p^{\uparrow} - \mathsf{var}_v^{\Uparrow})$
  $\triangleright\ (\widehat{x}_v; \widehat{\mathsf{var}}_v) \leftarrow \mathsf{CombineEstimates}((z_v^{\Uparrow}; \mathsf{var}_v^{\Uparrow}), (z_v^{\downarrow}; \mathsf{var}_v^{\downarrow}))$
  $\triangleright\ (z_v^{\Downarrow}; \mathsf{var}_v^{\Downarrow}) \leftarrow \mathsf{CombineEstimates}((x_v; \mathsf{var}_v), (z_v^{\downarrow}; \mathsf{var}_v^{\downarrow}))$
return $(\widehat{x}_v; \widehat{\mathsf{var}}_v)_{v \in T}$

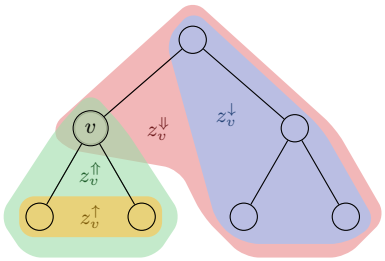


Figure 3: Intermediate variables in Algorithm 2.

The second pass proceeds top-down and computes for each non-root node v, a estimate $z_v^{\downarrow}$, which is an optimal linear unbiased estimate of c_v using only the noisy counts x_u for all u that are *not* in the sub-tree rooted at v. The estimates $z_v^{\Uparrow}$ and $z_v^{\downarrow}$ are then combined to obtain $\widehat{x}_v$, which is the optimal linear unbiased estimate of c_v using all the estimates x_u for all u in T. To assist with the recursive procedure, it also computes an estimate $z_v^{\Downarrow}$, which is the optimal linear unbiased estimate of c_v using only the noisy counts x_u for all u that are not strictly in the sub-tree under v (i.e., includes x_v). See Figure 3.

Algorithm 2 not only reduces the variance of each estimate but also achieves the smallest possible variance among all unbiased linear estimators, simultaneously for all nodes.

Theorem 8 (Optimality of Post-Processing). *For every $v \in T$, $\widehat{x}_v$ is the best linear unbiased estimator (BLUE) of the count c_v and has variance $\widehat{\mathsf{var}}_v$. In particular, $(\widehat{x}_v)_{v \in T}$ minimizes $\mathrm{RMSRE}_\tau(c_v, \widehat{x}_v)$ among all linear unbiased estimators, for all $v \in T$.*

This extends the results from [11, 12], 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 $(\widehat{x}_v)_{v \in T}$ 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 9. *For any T and $(x_v; \mathsf{var}_v)_{v \in T}$, let $(\widehat{x}_v; \widehat{\mathsf{var}}_v)_{v \in T}$ be the output of Algorithm 2. Then, the following hold:*

- *(Consistency) For all internal v : $\widehat{x}_v = \sum_{u \in \mathrm{child}(v)} \widehat{x}_u$.*
- *(Weighted Root-to-Leaf Sum Preservation) For each leaf v, $\sum_{u \in \mathrm{anc}(v)} \frac{x_u}{\mathsf{var}_u} = \sum_{u \in \mathrm{anc}(v)} \frac{\widehat{x}_u}{\mathsf{var}_u}$, where $\mathrm{anc}(v)$ denotes the nodes on the path from v to r (inclusive).*

The proof of Lemma 9 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 provide the full proofs in Appendix A.

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) allows us to allocate the privacy budget arbitrarily among the nodes of the tree, and we can apply post-processing (Algorithm 2) 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 use a greedy iterative approach for optimizing the privacy budget allocation. Let k be the 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 $\mathrm{RMSRE}_\tau(T)$ when using ε/k additional privacy budget, and increase the privacy budget of that level by ε/k. The error $\mathrm{RMSRE}_\tau(T)$ can be computed directly using the variance $\widehat{\mathrm{var}}_v$ of each post-processed estimate $\widehat{x}_v$ as returned by Algorithm 2. We present the details of this greedy iterative approach in Appendix B.

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 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 [23] This dataset contains 15,995,634 clicks 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., time of click on an ad) and a potential subsequent conversion (purchase of the corresponding product) within a 30-day attribution window. We consider the attributes: `partner_id`, `product_country`, `device_type`, `product_age_group` & `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).

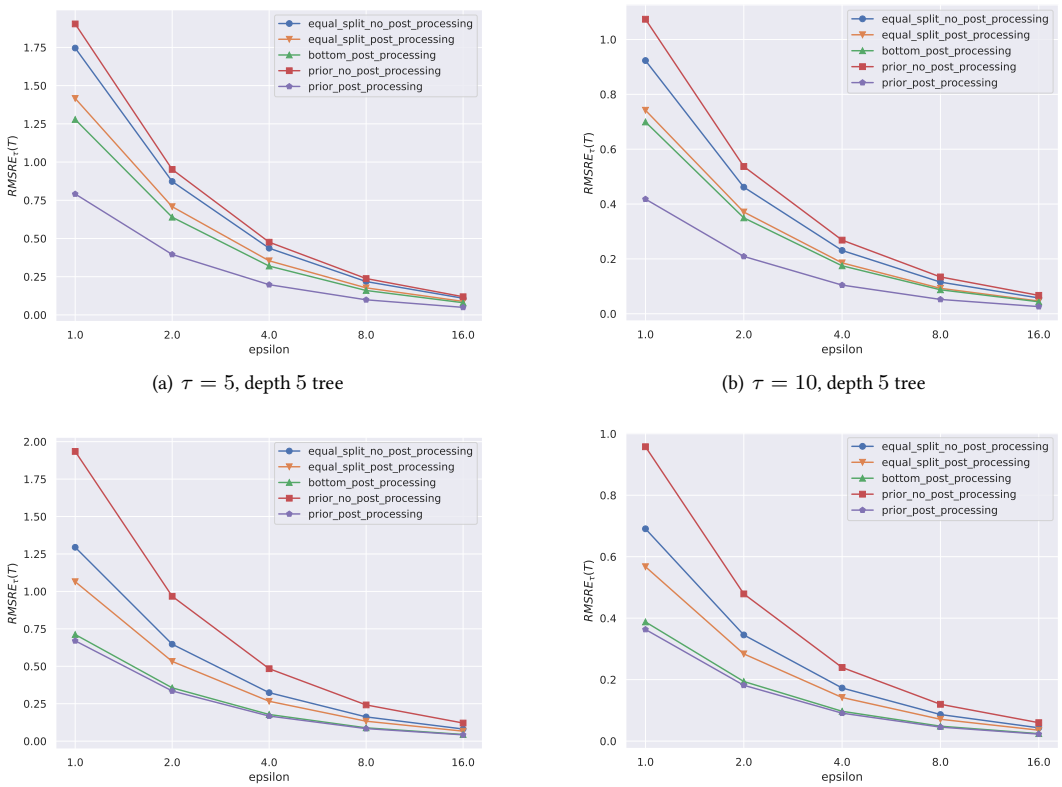
In Figure 4, we plot RMSRE_τ versus ε for various different methods evaluated on the later 45 days of data, namely,

- Equal privacy budget split across levels without any post-processing.
- Equal privacy budget split across levels with post-processing (Algorithm 2).
- All privacy budget on leaves with post-processing.
- Prior-based optimized privacy budget split across levels optimized for each `partner_id` without any post-processing.
- Prior-based optimized privacy budget split across levels optimized for each `partner_id` with post-processing (Algorithm 2).

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 $\varepsilon = 1$ 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 $\varepsilon = 4$ and the depth 5 tree, $\mathrm{RMSRE}_{10} \approx 0.1$, and for the depth 4 tree, $\mathrm{RMSRE}_5 \approx 0.17$.

Criteo Attribution Modeling for Bidding (CAMB) Dataset [24] It consists of ∼16M impressions from 30 days of Criteo live traffic. We consider last-touch attribution with impression attributes: `campaign`, categorical features `cat1`, `cat8`, and a discretization of conversion delay, i.e., the gap between `conversion_timestamp` and (impression) `timestamp`. As for CSSCL, we consider two discretizations for the difference, with two tree depths. Figure 5 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 $\varepsilon = 4$ and the depth 4 tree, $\mathrm{RMSRE}_{10} \approx 0.19$, and for the depth 3 tree (omitting attribute `cat8`), $\mathrm{RMSRE}_5 \approx 0.20$ and $\mathrm{RMSRE}_{10} \approx 0.12$.

Our prior-based budgeting with post-processing method equals or outperforms all other approaches in each setting (Figures 4 and 5).



(a) $\tau = 5$, depth 5 tree

(b) $\tau = 10$, depth 5 tree

(c) $\tau = 5$, depth 4 tree

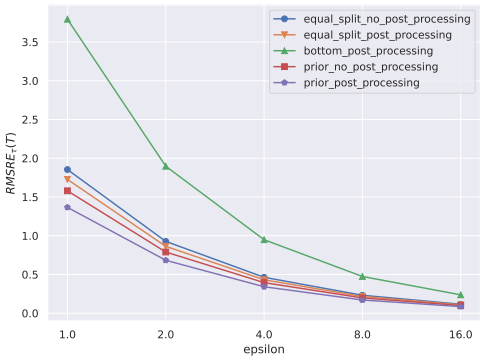
(d) $\tau = 10$, depth 4 tree

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

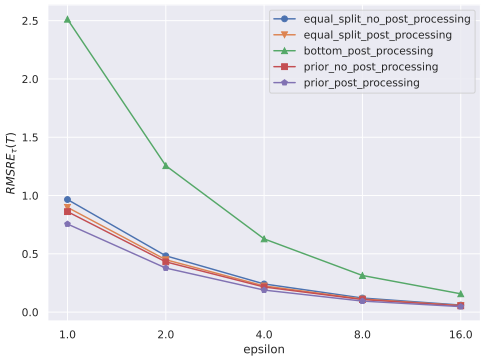
7. Conclusion and Future Directions

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.

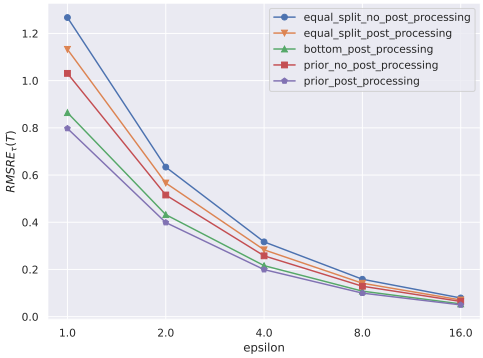
Another natural direction is to extend the consistency enforcement algorithm to ensure monotonicity (i.e., that the output estimate for a node of the tree is at least the output estimate for any of its children), and non-negativity. Our privacy budgeting method optimizes for one privacy parameter *per-level* 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 interesting direction to study privacy budgeting under approximate DP [25]. Our work considered the hierarchical query model; a natural direction is to optimize the *direct query model* [26]. Finally, the Attribution Reporting API also offers *event-level reports* [27]. It would be interesting to see if these could be also used to further improve the accuracy for hierarchical queries.



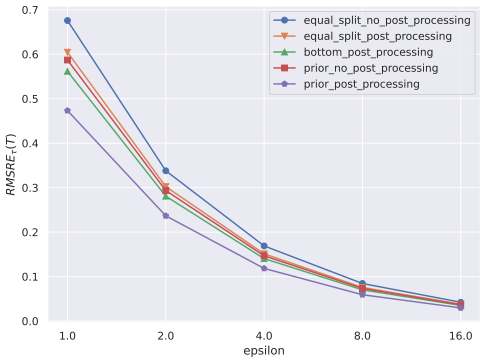
(a) $\tau = 5$, depth 4 tree



(b) $\tau = 10$, depth 4 tree



(c) $\tau = 5$, depth 3 tree



(d) $\tau = 10$, depth 3 tree

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

References

[1] Wikipedia, HTTP cookie: third-party cookie, https://en.wikipedia.org/wiki/HTTP_cookie#Third-party_cookie, 2023.

[2] Q. Lu, S. Pan, L. Wang, J. Pan, F. Wan, H. Yang, A practical framework of conversion rate prediction for online display advertising, in: AdKDD, 2017.

[3] Y. Choi, M. Kwon, Y. Park, J. Oh, S. Kim, Delayed feedback model with negative binomial regression for multiple conversions, in: AdKDD, 2020.

[4] Y. Qiu, N. Tziortziotis, M. Hue, M. Vazirgiannis, Predicting conversions in display advertising based on URL embeddings, in: AdKDD, 2020.

[5] T. Gu, K. Kuang, H. Zhu, J. Li, Z. Dong, W. Hu, Z. Li, X. He, Y. Liu, Estimating true post-click conversion via group-stratified counterfactual inference, in: AdKDD, 2021.

[6] J. Wilander, Full Third-Party Cookie Blocking and More, 2020. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[7] M. Wood, 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/.

[8] J. Schuh, Building a more private web: A path towards making third party cookies obsolete, 2020. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html.

[9] M. Nalpas, A. White, Attribution Reporting, 2021. https://developer.chrome.com/en/docs/privacy-sandbox/attribution-reporting/.

[10] Attribution reporting: Aggregatable reports API, 2021. https://developer.android.com/design-for-safety/privacy-sandbox/attribution#aggregatable-reports-api.

[11] M. Hay, V. Rastogi, G. Miklau, D. Suciu, Boosting the accuracy of differentially-private histograms through consistency, in: VLDB, 2010, pp. 1021–1032.

[12] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, T. Yu, Differentially private spatial decompositions, in: ICDE, 2012, pp. 20–31.

[13] C. Dwork, F. McSherry, K. Nissim, A. D. Smith, Calibrating noise to sensitivity in private data analysis, in: TCC, 2006, pp. 265–284.

[14] K. Chaudhuri, C. Monteleoni, A. D. Sarwate, Differentially private empirical risk minimization., JMLR 12 (2011).

[15] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, C. Zhang, Deep learning with label differential privacy, in: NeurIPS, 2021, pp. 27131–27145.

[16] H. Esfandiari, V. Mirrokni, U. Syed, S. Vassilvitskii, Label differential privacy via clustering, in: AISTATS, 2022, pp. 7055–7075.

[17] B. Ghazi, P. Kamath, R. Kumar, E. Leeman, P. Manurangsi, A. Varadarajan, C. Zhang, Regression with label differential privacy, in: ICLR, 2023. URL: https://openreview.net/pdf?id=h9O0wsmL-cT.

[18] C. Dwork, A. Roth, et al., The algorithmic foundations of differential privacy, Foundations and Trends® in Theoretical Computer Science 9 (2014) 211–407.

[19] Attribution Reporting Draft Community Group Report, 2021. https://wicg.github.io/attribution-reporting-api/.

[20] Attribution reporting: Trigger filters, 2021. https://developer.android.com/design-for-safety/privacy-sandbox/attribution#trigger-filters.

[21] Set up Aggregation Service for Aggregatable Reports: Collect and Batch Aggregatable Reports, 2021. https://github.com/privacysandbox/aggregation-service#collect-and-batch-aggregatable-reports.

[22] A. Nadan, A. White, A. Cucu, M. Nalpas, Z. Mastromatto, Experiment with summary report design decisions, 2022. https://developer.chrome.com/docs/privacy-sandbox/summary-reports/design-decisions/.

[23] M. Tallis, P. Yadav, Reacting to variations in product demand: An application for conversion rate (CR) prediction in sponsored search, in: IEEE BigData, 2018, pp. 1856–1864.

[24] J. Meynet, Criteo attribution modeling for bidding dataset, 2017. https://ailab.criteo.com/criteo-attribution-modeling-bidding-dataset/.

[25] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, M. Naor, Our data, ourselves: Privacy via distributed noise generation, in: EUROCRYPT, 2006, pp. 486–503.

[26] Privacy Sandbox Aggregation Service: Query Models, 2021. https://github.com/google/privacy-sandbox-aggregation-service#query-models.

[27] Attribution Reporting: Event-level Reports, 2021. https://developer.android.com/design-for-safety/privacy-sandbox/attribution#event-level-reports.

[28] S. Silvey, Statistical Inference, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis, 1975.

A. Proof of Least Squares Optimality

We provide the proof of Theorem 8, starting with the proof of Lemma 9, restated below for convenience.

Lemma 9. *For any T and $(x_v;\mathsf{var}_v)_{v\in T}$, let $(\widehat{x}_v;\widehat{\mathsf{var}}_v)_{v\in T}$ be the output of Algorithm 2. Then, the following hold:*

- *(Consistency) For all internal v : $\widehat{x}_v = \sum_{u\in\mathrm{child}(v)} \widehat{x}_u$.*
- *(Weighted Root-to-Leaf Sum Preservation) For each leaf v, $\sum_{u\in\mathrm{anc}(v)} \frac{x_u}{\mathsf{var}_u} = \sum_{u\in\mathrm{anc}(v)} \frac{\widehat{x}_u}{\widehat{\mathsf{var}}_u}$, where $\mathrm{anc}(v)$ denotes the nodes on the path from v to r (inclusive).*

Proof. We will prove this by (strong) induction on the number of nodes in T. The base case where T has a single node is immediate as the bottom-up and top-down passes are vacuous. We use the inductive hypothesis that both *consistency* and *weighted root-to-leaf sum preservation* properties hold for the output of Algorithm 2 on any tree of at most m nodes for $m \in \mathbb{N}$.

Let T be any tree with $m+1$ nodes. For any input $(x_v;\mathsf{var}_v)_{v\in T}$, let $(z_v^{\uparrow};\mathsf{var}_v^{\uparrow})$, $(z_v^{\Uparrow};\mathsf{var}_v^{\Uparrow})$, $(z_v^{\downarrow};\mathsf{var}_v^{\downarrow})$, $(z_v^{\Downarrow};\mathsf{var}_v^{\Downarrow})$, $(\widehat{x}_v;\widehat{\mathsf{var}}_v)$ be the values computed by Algorithm 2 for every $v\in T$

Let v^* be any internal node whose children are all leaves, and T' denote the tree with its children removed. For all $v\in T'$, define

$$(x'_v;\mathsf{var}'_v) = \begin{cases} (x_v;\mathsf{var}_v) & \text{if } v\neq v^*,\\ (z_v^{\Uparrow};\mathsf{var}_v^{\Uparrow}) & \text{if } v = v^*. \end{cases} \tag{1}$$

Similar to above, let $(z'_v{}^{\uparrow};\mathsf{var}'_v{}^{\uparrow})$, $(z'_v{}^{\Uparrow};\mathsf{var}'_v{}^{\Uparrow})$, $(z'_v{}^{\downarrow};\mathsf{var}'_v{}^{\downarrow})$, $(z'_v{}^{\Downarrow};\mathsf{var}'_v{}^{\Downarrow})$, $(\widehat{x}'_v;\widehat{\mathsf{var}}'_v)$ be as defined in Algorithm 2 when run on the tree T' with input $(x'_v;\mathsf{var}'_v)_{v\in T'}$. From the inductive hypothesis, for all internal nodes $v\in T'$, we have

$$\widehat{x}'_v = \sum_{u\in\mathrm{child}(v)} \widehat{x}'_u, \tag{2}$$

and for every leaf $v\in T'$, we have

$$\sum_{u\in\mathrm{anc}(v)} \frac{x'_u}{\mathsf{var}'_u} = \sum_{u\in\mathrm{anc}(v)} \frac{\widehat{x}'_u}{\mathsf{var}'_u}. \tag{3}$$

When we run Algorithm 2 on T, after setting $z_{v^*}^{\Uparrow}$ and $\mathsf{var}_{v^*}^{\uparrow}$, the rest of the bottom-up pass is exactly the same as that of the run on T'. Similarly, the top-down pass of T' is the same as that of T except that in T we also set the values of $\widehat{x}_u$ and $\widehat{\mathsf{var}}_u$ for all $u\in\mathrm{child}(v^*)$. Therefore,

$$\widehat{x}_v = \begin{cases} \widehat{x}'_v & \text{if } v\notin\mathrm{child}(v^*),\\ \frac{\mathsf{var}_v^{\downarrow}\cdot x_v + \mathsf{var}_v\cdot z_v^{\downarrow}}{\mathsf{var}_v^{\downarrow}+\mathsf{var}_v} & \text{if } v\in\mathrm{child}(v^*), \end{cases} \tag{4}$$

where

$$(z_v^{\downarrow};\mathsf{var}_v^{\downarrow}) = (\widehat{x}_{v^*} - z_{v^*}^{\uparrow} + x_v;\ \widehat{\mathsf{var}}_{v^*} + \mathsf{var}_{v^*}^{\uparrow} - \mathsf{var}_v).$$

Here we used the observation that for all leaves v of T' (including v^*), it holds that $(z'_v{}^{\Uparrow};\mathsf{var}'_v{}^{\Uparrow}) = (x_v;\mathsf{var}_v)$ and hence $(z'_v{}^{\Downarrow};\mathsf{var}'_v{}^{\Downarrow}) = (\widehat{x}'_v;\widehat{\mathsf{var}}'_v) = (\widehat{x}_v;\widehat{\mathsf{var}}_v)$.

(Consistency) For $v\neq v^*$, Equation (4) implies that the LHS and RHS of the desired consistency condition is exactly the same as that of Equation (2). So it only remains to show consistency for $v=v^*$. First, we simplify Equation (4) by noting that $\mathsf{var}_v^{\downarrow}+\mathsf{var}_v = \mathsf{var}_{v^*}^{\Downarrow}+\mathsf{var}_{v^*}^{\uparrow}$ for all $v\in\mathrm{child}(v^*)$, and hence for $v\in\mathrm{child}(v^*)$ we have

$$\begin{aligned}\widehat{x}_v &= x_v + \frac{\mathsf{var}_v}{\mathsf{var}_{v^*}^{\Downarrow}+\mathsf{var}_{v^*}^{\uparrow}}\left(z_{v^*}^{\Downarrow} - z_{v^*}^{\uparrow}\right)\\ &= x_v + \frac{\mathsf{var}_v}{\mathsf{var}_v^{\uparrow}}\left(\frac{\mathsf{var}_{v^*}^{\uparrow} z_{v^*}^{\Downarrow} + \mathsf{var}_{v^*}^{\Downarrow} z_{v^*}^{\uparrow}}{\mathsf{var}_{v^*}^{\Downarrow}+\mathsf{var}_{v^*}^{\uparrow}} - z_{v^*}^{\uparrow}\right)\\ &= x_v + \frac{\mathsf{var}_v}{\mathsf{var}_{v^*}^{\uparrow}}\left(\widehat{x}_{v^*} - z_{v^*}^{\uparrow}\right).\end{aligned}$$

Hence, we have

$$\begin{aligned}&\sum_{u\in\mathrm{child}(v^*)} \widehat{x}_u\\ &= \sum_{u\in\mathrm{child}(v^*)}\left(x_u + \frac{\mathsf{var}_u}{\mathsf{var}_{v^*}^{\uparrow}}\cdot(\widehat{x}_{v^*} - z_{v^*}^{\uparrow})\right)\\ &= \widehat{x}_{v^*}\end{aligned}$$

where we use that $z_{v^*}^{\uparrow} = \sum_{u\in\mathrm{child}(v^*)} x_u$. Thus, the consistency condition holds for every internal node $v\in T$ as desired.

(Weighted Root-to-Leaf Sum Preservation) Again, for $v \neq v^*$, Equation (4) and Equation (3) imply the weighted root-to-leaf sum preservation property for all leaves $v \notin \text{child}(v^*)$. Meanwhile, for $v \in \text{child}(v^*)$, we have

$$\begin{aligned}
&\sum_{u \in \text{anc}(v)} \frac{\widehat{x}_u}{\mathsf{var}_u} \\
&= \frac{\widehat{x}_v}{\mathsf{var}_v} + \frac{\widehat{x}_{v^*}}{\mathsf{var}_{v^*}} - \frac{\widehat{x}_{v^*}}{\mathsf{var}^{\Uparrow}_{v^*}} + \sum_{u \in \text{anc}(v^*)} \frac{\widehat{x}'_u}{\mathsf{var}'_u} \\
&\overset{(3)}{=} \frac{\widehat{x}_v}{\mathsf{var}_v} + \frac{\widehat{x}_{v^*}}{\mathsf{var}_{v^*}} - \frac{\widehat{x}_{v^*}}{\mathsf{var}'_{v^*}} + \sum_{u \in \text{anc}(v^*)} \frac{x'_u}{\mathsf{var}'_u} \\
&\overset{(1)}{=} \frac{\widehat{x}_v}{\mathsf{var}_v} + \frac{\widehat{x}_{v^*}}{\mathsf{var}_{v^*}} - \frac{\widehat{x}_{v^*}}{\mathsf{var}^{\Uparrow}_{v^*}} - \frac{x_{v^*}}{\mathsf{var}_{v^*}} + \frac{z^{\Uparrow}_{v^*}}{\mathsf{var}^{\Uparrow}_{v^*}} \\
&\quad + \sum_{u \in \text{anc}(v^*)} \frac{x_u}{\mathsf{var}_u}. \qquad (5)
\end{aligned}$$

Recall that

$$\begin{aligned}
\mathsf{var}^{\Uparrow}_{v^*} &= \frac{\mathsf{var}_{v^*} \cdot \mathsf{var}^{\uparrow}_{v^*}}{\mathsf{var}_{v^*} + \mathsf{var}^{\uparrow}_{v^*}}, \\
\widehat{x}_v &= x_v + \frac{\mathsf{var}_v}{\mathsf{var}^{\uparrow}_{v^*}} \left(\widehat{x}_{v^*} - z^{\uparrow}_{v^*} \right), \\
z^{\Uparrow}_{v^*} &= \mathsf{var}^{\Uparrow}_{v^*} \cdot \left(\frac{x_{v^*}}{\mathsf{var}_{v^*}} + \frac{z^{\uparrow}_{v^*}}{\mathsf{var}^{\uparrow}_{v^*}} \right).
\end{aligned}$$

Hence the first five terms in Equation (5) can be written as

$$\begin{aligned}
&\frac{\widehat{x}_v}{\mathsf{var}_v} + \frac{\widehat{x}_{v^*}}{\mathsf{var}_{v^*}} - \frac{\widehat{x}_{v^*}}{\mathsf{var}^{\Uparrow}_{v^*}} - \frac{x_{v^*}}{\mathsf{var}_{v^*}} + \frac{z^{\Uparrow}_{v^*}}{\mathsf{var}^{\Uparrow}_{v^*}} \\
&= \frac{1}{\mathsf{var}_v} \left(x_v + \frac{\mathsf{var}_v}{\mathsf{var}^{\uparrow}_{v^*}} \cdot (\widehat{x}_{v^*} - z^{\uparrow}_{v^*}) \right) \\
&\quad + \frac{\widehat{x}_{v^*}}{\mathsf{var}_{v^*}} - \frac{\mathsf{var}_{v^*} + \mathsf{var}^{\uparrow}_{v^*}}{\mathsf{var}_{v^*} \cdot \mathsf{var}^{\uparrow}_{v^*}} \cdot \widehat{x}_{v^*} - \frac{x_{v^*}}{\mathsf{var}_{v^*}} \\
&\quad + \frac{x_{v^*}}{\mathsf{var}_{v^*}} + \frac{z^{\uparrow}_{v^*}}{\mathsf{var}^{\uparrow}_{v^*}} \\
&= \frac{x_v}{\mathsf{var}_v}.
\end{aligned}$$

Thus, we get that

$$\sum_{u \in \text{anc}(v)} \frac{\widehat{x}_u}{\mathsf{var}_u} = \sum_{u \in \text{anc}(v)} \frac{x_u}{\mathsf{var}_u},$$

holds for all $v \in T$. This completes our proof. □

We additionally need the Gauss–Markov theorem stated below for noise with non-uniform diagonal covariance.

Theorem 10 (Gauss–Markov (see e.g., [28])). *Fix $A \in \mathbb{R}^{n \times d}$. For an unknown $\theta \in \mathbb{R}^d$, suppose we observe $x = A\theta + e$ where $e \in \mathbb{R}^d$ is drawn such that e_i's are independent with $\mathbb{E}\, e_i = 0$ and variance var_i respectively. Then, the least squares estimator $\hat{\theta}$ defined as the minimizer of $\sum_i (x_i - (A\hat{\theta})_i)^2 / \mathsf{var}_i$, is the best unbiased linear estimator (BLUE), namely, for all $\alpha \in \mathbb{R}^d$ it holds that $\mathsf{Var}(\alpha^T \hat{\theta})$ is the smallest among all linear unbiased estimators of $\alpha^\top \theta$.*

Finally, we prove Theorem 8 (restated below) using Lemma 9 and Theorem 10.

Theorem 8 (Optimality of Post-Processing). *For every $v \in T$, $\widehat{x}_v$ is the best linear unbiased estimator (BLUE) of the count c_v and has variance $\widehat{\mathsf{var}}_v$. In particular, $(\widehat{x}_v)_{v \in T}$ minimizes $\text{RMSRE}_\tau(c_v, \widehat{x}_v)$ among all linear unbiased estimators, for all $v \in T$.*

Proof. Corresponding to any tree T, we can associate the matrix $A \in \mathbb{R}^{n \times d}$, where n is the number of nodes (both internal and leaf nodes), and d is the number of leaf nodes, such that $A_{u,v}$ is 1 if leaf v is either equal to or a descendant of u and 0 otherwise. The estimated counts are given as $x = A\theta + e$ where $\theta \in \mathbb{R}^d$ is such that θ_v is the true count for leaf v. We have that $\hat{\theta}$ is a least squares estimator minimizing $f(\tilde{\theta}) := \sum_u (x_u - (A\tilde{\theta})_u)^2 / \mathsf{var}_u$ if and only if it satisfies $\nabla f(\hat{\theta}) = 0$ (due to convexity of squared loss). Setting the derivative w.r.t. $\hat{\theta}_v$ equal to 0, implies that for each leaf v,

$$\begin{aligned}
\sum_u \frac{x_u - (A\hat{\theta})_u}{\mathsf{var}_u} \cdot A_{u,v} &= 0 \\
\iff \sum_{u \in \text{anc}(v)} \frac{x_u}{\mathsf{var}_u} &= \sum_{u \in \text{anc}(v)} \frac{(A\hat{\theta})_u}{\mathsf{var}_u}.
\end{aligned}$$

Thus, the best linear unbiased estimate for $(A\theta)_u$ is given by $\widehat{x}_u = (A\hat{\theta})_u$, which clearly satisfies *consistency* and as shown above also satisfies the *weighted root-to-leaf sum preservation* properties of Lemma 9. Conversely, if $(\widehat{x}_u)_u$ satisfies consistency, then it must be of the form $A\tilde{\theta}$ for $\tilde{\theta}_u = \widehat{x}_u$ for all leaves u, and if it satisfies the *weighted root-to-leaf sum preservation* property, then as shown above $\tilde{\theta}$ must be the least-squares estimator.

Thus, from Lemma 9, we have that the $(\widehat{x}_u)_u$ returned by Algorithm 2 satisfies the two properties, we have that each $\widehat{x}_u$ is the BLUE for the corresponding true count c_u by Theorem 10.

Finally, to see that $\widehat{\mathsf{var}}_v$ is indeed the variance of estimate $\widehat{x}_v$, we recursively show that $\mathsf{var}^{\uparrow}_v$, $\mathsf{var}^{\Uparrow}_v$, $\mathsf{var}^{\downarrow}_v$, $\mathsf{var}^{\Downarrow}_v$ are the variances corresponding to $z^{\uparrow}_v$, $z^{\Uparrow}_v$, $z^{\downarrow}_v$, $z^{\Downarrow}_v$ respectively. The key property to verify is that each z quantity involves a linear combination of estimates which depend on noisy counts of disjoint parts of the tree and hence are independent. Thus, we can repeatedly use the property that for independent drawn X and Y, it holds that $\mathsf{Var}(\alpha X + \beta Y) = \alpha^2 \mathsf{Var}(X) + \beta^2 \mathsf{Var}(Y)$. □

B. Greedy Iterative Budgeting

Algorithm 3 presents the greedy iterative approach we used for optimizing the privacy budget allocation using true counts c, as alluded to in Section 5. The high level idea is as follows. We choose a parameter k to be a number of phases (e.g., $k = 20$) corresponding to the granularity of the allocation.

Initially, allocate an infinitesimal privacy budget to each level; this is done so that we can use Algorithm 2 to compute $\widehat{\mathsf{var}}_v$ for each node of the tree; we need this infinitesimal allocation because Algorithm 2 as written does not allow var_v to be ∞ for any v.[4]

We divide the (remaining) privacy budget of $\widehat{\varepsilon}$ into k units of size $\widehat{\varepsilon}/k$. In each of k phases, we consider adding $\widehat{\varepsilon}/k$ budget to all nodes at level i, choosing an i that results in the lowest $\mathrm{RMSRE}_\tau(T)$, which can be computed using Algorithm 2; for any budgeting sequence $(\varepsilon_0, \ldots, \varepsilon_d)$, we set var_v to be the variance of $\mathrm{DLap}(1/\varepsilon_i)$ for all nodes v in level L_i. Observe that $\mathrm{RMSRE}_\tau(T)$ can be computed directly using the variance $\widehat{\mathsf{var}}_v$ of each post-processed estimate $\widehat{x}_v$ as returned by Algorithm 2 since

$$\mathrm{RMSRE}_\tau(c_v, \widehat{x}_v) \;=\; \sqrt{\frac{\widehat{\mathsf{var}}_v}{\max(\tau, c_v)^2}}\,.$$

Algorithm 3 Privacy budgeting via greedy iterations

Params: Tree T, with levels $L_0, L_1, \ldots, L_d$.
Input: Total privacy budget ε, Number of phases k
Output: Budget split $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_d$ such that $\sum_i \varepsilon_i = \varepsilon$.

Choose an infinitesimal γ, e.g., $\gamma \leftarrow 10^{-5}$.
for $i = 0, \ldots, d$ **do**
  $\varepsilon_i \leftarrow \gamma \cdot \varepsilon/d$
$\widehat{\varepsilon} \leftarrow (1-\gamma)\varepsilon$ (remaining privacy budget)
for $j = 1, \ldots, k$ **do**
  for $i = 0, \ldots, d$ **do**
    $R_i \leftarrow \mathrm{RMSRE}_\tau(T)$ using privacy budget split of $(\varepsilon_0, \ldots, \varepsilon_i + \widehat{\varepsilon}/k, \ldots, \varepsilon_d)$
    (computed using $(\widehat{\mathsf{var}}_v)_{v \in T}$ from Algorithm 2)
  $\ell \leftarrow \mathrm{argmin}_i\, R_i$
  $\varepsilon_\ell \leftarrow \varepsilon_\ell + \widehat{\varepsilon}/k$
return $(\varepsilon_0, \ldots, \varepsilon_d)$

[4] While it is possible to modify Algorithm 2 to support $\mathsf{var}_v = \infty$ for certain subsets of the nodes, we avoid doing so for retaining clarity.