DP-SIPS: A simpler, more scalable mechanism for differentially private partition selection

Marika Swanberg*
Boston University
Department of Computer Science
marikas@bu.edu

Damien Desfontaines
Tumult Labs
damien@desfontain.es

Samuel Haney
Tumult Labs
sam.haney@tmlt.io

ABSTRACT

Partition selection, or set union, is an important primitive in differentially private mechanism design: in a database where each user contributes a list of items, the goal is to publish as many of these items as possible under differential privacy.

In this work, we present a novel mechanism for differentially private partition selection. This mechanism, which we call DP-SIPS, is very simple: it consists of iterating the naive algorithm over the data set multiple times, removing the released partitions from the data set while increasing the privacy budget at each step. This approach preserves the scalability benefits of the naive mechanism, yet its utility compares favorably to more complex approaches developed in prior work.

KEYWORDS

Differential privacy, partition selection, scalable algorithms

1 INTRODUCTION

Group-level aggregation is a fundamental building block for many data analysis tasks. For example, doctors may want to understand mortality rates for patients with different underlying conditions, and search engine developers may want to study the frequency of different search queries that users are making.

With increased awareness of the risks of data disclosure, increasingly data analysts are using differentially private mechanisms to answer queries on sensitive data sets. In both of the given examples, the aggregate values (e.g., the mortality rates, or the frequencies of search queries) are sensitive, but also the groups themselves are sensitive. Revealing that *some* user in the data set has a particular disease or made a particular search query could *itself* violate privacy—even if it is never revealed *who* that person is. Additionally, the set of groups may be *a priori* unknown or impractical to enumerate, like in vocabulary extraction [11] or the private release of search queries [14].

In order to privately do a group-level aggregation in such settings, one must first privately compute the set of groups represented in the data set—ideally, releasing as many as possible while still maintaining privacy. More formally, the mechanism M gets a data set $x = (W_1, \ldots, W_N)$ where each user i has a list of items W_i and M differentially privately releases a subset $S \subseteq \cup_i W_i$ of the partitions that is as large as possible.

We study this problem, called *differentially private partition selection* (also called key selection or set union), with a focus on approaches that can be incorporated into general-purpose differentially private tooling. When designing such frameworks, scalability is of paramount importance [1, 2, 21]: underlying mechanisms must be able to run even when the input or the output is too large to fit on a single machine. In such cases, the computation itself must also be parallelizable, so it can run across multiple machines and avoid unacceptably long running times.

Differentially private partition selection mechanisms have been proposed as early as 2009 [14], but recent work has shed a new light on this problem, and proposed alternative approaches that bring significant utility gains [6, 11]. To obtain these utility improvements, these newer mechanisms use a *greedy* approach: each user considers what items have been contributed by previous users so far, and “chooses” which items to contribute according to a *policy*, chosen carefully to maintain a sensitivity bound. Unfortunately, these mechanisms do not scale: each user chooses their contribution based on the contribution of all previous users, so the data has to be processed one user after another, and the overall algorithm cannot be parallelized. This intuition can be verified experimentally: we show that such algorithms eventually time out or run out of memory when the input is very large, even on clusters of multiple machines. Because these greedy algorithms do not scale, differential privacy tools that need to handle large datasets cannot use these smarter approaches, and instead must rely on the naive algorithm and its underwhelming utility [9, 10, 15].

This raises a natural question: can we achieve the utility benefits of policy-based approaches, while preserving the scalability of more naive approaches? In this work, we introduce a new approach that combines both benefits: DP-SIPS, short for *scalable, iterative partition selection.*

DP-SIPS relies on a simple idea: rather than having to process the data of each user sequentially, it runs the naive, massively-parallelizable algorithm multiple times, splitting the privacy budget between each step and removing items that were previously discovered. It uses a minuscule amount of privacy budget to publish and then remove the “heavy hitters”–those which have an enormous amount of weight in the histogram–and allocates the majority of the budget to discovering items that remain after heavy hitters are removed from the dataset. On skewed datasets, the removal of heavy hitters allows users to allocate proportionally more weight in subsequent histograms to less-frequent items while the increased privacy budget on later iterations also *lowers* the threshold for releasing an item.

*Work done while at Tumult Labs

Proceedings on Privacy Enhancing Technologies 2023(4), 257–268

https://doi.org/10.56553/popets-2023-0109

As we show experimentally on multiple real-world and synthetic datasets, this mechanism has a similar utility to greedy approaches, but scales horizontally: increasing the number of cluster nodes significantly reduces runtime. This makes it a suitable choice for implementation in general-purpose DP infrastructure with high scalability requirements.

The rest of this paper is organized as follows:

- In Section 2, we formally define the problem and the building blocks we use for DP-SIPS and its privacy accounting.
- In Section 3, we detail existing approaches to differentially private partition selection.
- In Section 4, we introduce our algorithm and the proof of its privacy guarantees.
- In Section 5, we report on the experimental evaluation of DP-SIPS.
- Finally, in Section 6, we discuss our results, report on some unsuccessful approaches that we tried, and outline directions for future work.

2 PRELIMINARIES

A data set $x = (W_1, \dots, W_N)$ contains a set of user lists $W_i \in \mathcal{U}^*$. We refer to elements in $\mathcal{U}$ as *items* or *partitions*, and define partition selection (also called key selection or set union) as follows:

Definition 1 (Private Partition Selection Problem). *Given a (possibly unbounded) universe $\mathcal{U}$ of items and a data set $x = (W_1, \dots, W_N)$ of user lists $W_i \in \mathcal{U}^*$, an algorithm $\mathcal{M}$ solves the* private partition selection problem *if it is differentially private, and $\mathcal{M}$ outputs a set $S \subseteq \cup_i W_i$.*

We begin by presenting the standard notion of differential privacy. Two data sets x, x' are *neighbors* if they differ on one user's list: $x = x' \cup W_{i^*}$. Informally, differential privacy requires that an algorithm's output is distributed similarly on every pair of neighboring data sets.

Definition 2 (Differential Privacy [7, 8]). *A randomized algorithm $\mathcal{M} : \mathcal{U}^* \to \mathcal{Y}$ is (ε, δ)-differentially private if for every pair of neighboring datasets $x, x' \in \mathcal{U}^*$ and for all subsets $Y \subseteq \mathcal{Y}$,*

$$\Pr[\mathcal{M}(x) \in Y] \le e^{\varepsilon} \cdot \Pr[\mathcal{M}(x') \in Y] + \delta.$$

A common variant of differential privacy, called zCDP, is useful for analyzing algorithms that sample noise from a Gaussian distribution (as ours will). The definition of zCDP uses the Rényi Divergence:

Definition 3 (Rényi Divergence). *Fix two probability distributions P and Q over a discrete domain S. Given a positive $\alpha \neq 1$, Rényi divergence of order α of distributions P and Q is*

$$D_\alpha(P||Q) = \frac{1}{1-\alpha} \log\left(\sum_{s\in S} P(s)^\alpha Q(s)^{1-\alpha}\right).$$

Definition 4 (ρ-zCDP [4]). *A randomized mechanism $\mathcal{M} : \mathcal{X}^* \to \mathcal{Y}$ satisfies ρ-zCDP if, for all $x, x' \in \mathcal{X}^*$ differing on a single entry,*

$$D_\alpha(M(x)||M(x')) \le \rho \cdot \alpha \quad \forall \alpha \in (1, \infty). \tag{1}$$

This definition can also be relaxed to *approximate* zCDP.

Definition 5 (Approximate zCDP [4]). *A randomized mechanism $M : \mathcal{X}^n \to \mathcal{Y}$ is δ-approximately ρ-zCDP if, for all $x, x' \in \mathcal{X}^n$ differing on a single entry, there exist events $E = E(M(x))$ and $E' = E'(M(x'))$ such that, for all $\alpha \in (1, \infty)$,*

$$D_\alpha(M(x)|_E||M(x')|_{E'}) \le \rho \cdot \alpha \quad \text{and} \tag{2}$$

$$D_\alpha(M(x')|_{E'}||M(x)|_E) \le \rho \cdot \alpha, \tag{3}$$

and $\Pr[E] \ge 1 - \delta$ and $\Pr[E'] \ge 1 - \delta$.

Approximate zCDP satisfies composition and post-processing properties.

Lemma 6 ([4], Lemma 8.2). *Let $M : \mathcal{X}^n \to \mathcal{Y}$ and $M' : \mathcal{X}^n \times \mathcal{Y} \to \mathcal{Z}$. Suppose M satisfies δ-approximate ρ-zCDP and for all $y \in \mathcal{Y}$, $M'(\cdot, y) : \mathcal{X}^n \to \mathcal{Z}$ satisfies δ'-approxmiate ρ'-zCDP. Define $M'' : \mathcal{X}^n \to \mathcal{Z}$ by $M''(x) = M'(x, M(x))$. Then, M'' satisfies $(\delta + \delta' - \delta \cdot \delta')$-approximate $(\rho + \rho')$-zCDP.*

Next we show a conversion from approximate zCDP to approximate DP. We rewrite the conversion lemma from [4] to be slightly more general (i.e. it uses an existing conversion from zCDP to approximate DP), and then apply the tight zCDP-to-approximate DP conversion given in [5]. Overall, this gives a tighter approximate zCDP-to-approximate DP conversion than what is stated in Lemma 8.8 of [4].

Lemma 7 (Generalized version of Lemma 8.8 from [4]). *Suppose we can show that every mechanism that satisfies ρ-zCDP must satisfy $\epsilon^*(\rho), \delta^*(\rho)$-approximate DP. That is, (ϵ^*, δ^*) is a function converting a (pure) zCDP guarantee to an approximate DP guarantee. Suppose $\mathcal{M} : \mathcal{X}^N \to \mathcal{Y}$ satisfies δ-approximate ρ-zCDP. Then, $\mathcal{M}$ satisfies $(\epsilon^*(\rho), \delta + (1-\delta)\delta^*(\rho))$-DP.*

Lemma 8 ([5], Proposition 7). *Suppose $\mathcal{M} : \mathcal{X}^N \to \mathcal{Y}$ satisfies ρ-zCDP. Then $\mathcal{M}$ satisfies (ϵ, δ)-approximate DP for any $\epsilon > 0$ and*

$$\delta = \inf_{\alpha \in (1,\infty)} \frac{\exp((\alpha-1)(\alpha \cdot \rho - \epsilon))}{\alpha - 1}\left(1 - \frac{1}{\alpha}\right)^\alpha. \tag{4}$$

Combining Lemma 7 and Lemma 8 gives us the following.

Corollary 9. *Suppose $\mathcal{M} : \mathcal{X}^N \to \mathcal{Y}$ satisfies δ-approximate ρ-zCDP. Then $\mathcal{M}$ satisfies $(\epsilon, \delta + (1-\delta)\delta')$-approximate DP for any $\epsilon > 0$ and*

$$\delta' = \inf_{\alpha \in (1,\infty)} \frac{\exp((\alpha-1)(\alpha \cdot \rho - \epsilon))}{\alpha - 1}\left(1 - \frac{1}{\alpha}\right)^\alpha. \tag{5}$$

A common primitive in building private algorithms, the Gaussian Mechanism, satisfies ρ-zCDP.

Definition 10 (Gaussian Distribution). *The Gaussian distribution with parameter σ and mean 0, denoted $\mathcal{N}(0, \sigma^2)$ is defined for all $\ell \in \mathbb{R}$ and has probability density*

$$h(\ell) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\ell^2}{2\sigma^2}}.$$

Definition 11 (ℓ_2-Sensitivity). *Let $f : \mathcal{U}^n \to \mathbb{R}^d$ be a function. Its ℓ_2-sensitivity is*

$$\Delta_f = \max_{\substack{x, x' \in \mathcal{U} \\ x, x' \, neighbors}} \|f(x) - f(x')\|_2.$$

Definition 12 (Gaussian Mechanism [4]). *Let $f : \mathcal{U}^n \to \mathbb{R}^d$ be a function with ℓ_2-sensitivity Δ_f. Then the Gaussian mechanism is the algorithm*

$$\mathcal{M}_f(x) = f(x) + (Z_1, \ldots, Z_d),$$

where $Z_i \sim \mathcal{N}\left(0, \frac{\Delta_f^2}{2\rho}\right)$. Algorithm $\mathcal{M}_f$ satisfies ρ-zCDP.

3 PRIOR APPROACHES TO PARTITION SELECTION

In this section we discuss three existing algorithms for differentially private partition selection. We begin with the naive algorithm, called Weighted Gaussian, in Section 3.1. In Section 3.2, we present Policy Gaussian [11] and Greedy updates Without sampling [6].

The algorithms all have three main steps: first, they compute a *weighted histogram*, which is simply a mapping from an item $u \in \mathcal{U}$ to a weight $H[u] \in \mathbb{R}$; second, they add calibrated noise to each item in the histogram; and lastly, they release items that are above some appropriately-chosen threshold T. The primary difference between the algorithms is in how they compute the weighted histogram. This high-level algorithm for private partition selection is described in Algorithm 1, which can be composed with different weighted histogram algorithms.

Algorithm 1 High-Level Partition Selection Algorithm

Input: Data set of user partitions $x = (W_1, \ldots, W_N)$
Weighted Histogram Algorithm `Weighted_Hist`
Threshold T
Noise distribution $\mathcal{D}$
Output: Partitions $S \subseteq \cup_i W_i$

Initialize empty set $S \leftarrow \{\}$
$H \leftarrow$ `Weighted_Hist`(x) ▷ Compute weighted histogram
for $u \in Supp(H)$ **do**
  $Z_u \sim \mathcal{D}$
  $\hat{H}[u] \leftarrow H[u] + Z_u$
  if $\hat{H}[u] \geq T$ **then**
    $S \leftarrow S \cup \{u\}$
return S

3.1 Baseline: Weighted Gaussian

The Weighted Gaussian algorithm (Algorithm 2) is one of the simplest and first algorithms for private partition selection [14]. To build a weighted histogram, the algorithm first pre-processes the data set by removing any duplicates within a user's set and truncating each user's set to have at most Δ_0 items. Then, the users compute a histogram with bounded ℓ_2-sensitivity as follows: each user i updates the weight $H[u]$ for each of the items u in their set W_i with the following rule:

$$H[u] \leftarrow H[u] + \frac{1}{\sqrt{|W_i|}}.$$

The resulting weighted histogram has an ℓ_2-sensitivity of 1, so it can be composed with the high-level algorithm using calibrated Gaussian noise and an appropriate threshold to ensure that the overall algorithm satisfies δ-approximate ρ-zCDP. In the statement of Algorithm 2, $\Phi(\cdot)$ is used to denote the cumulative density function of the standard Gaussian distribution and $\Phi^{-1}(\cdot)$ is its inverse.

Algorithm 2 Weighted Gaussian

Input: Data set $x = (W_1, \ldots, W_N)$
Privacy parameters (ρ, δ)
Maximum per-user contribution Δ_0
Output: Partitions $S \subseteq \cup_i W_i$

Initialize empty histogram $H \leftarrow \{\}$
Initialize empty set $S \leftarrow \{\}$
for $i = 1, \ldots, N$ **do**
  $W_i \leftarrow$ get rid of duplicate items from W_i
  $\overline{W_i} \leftarrow$ uniformly sample at most Δ_0 items from W_i
  for $u \in \overline{W}_i$ **do**
    $H[u] \leftarrow H[u] + \frac{1}{\sqrt{|\overline{W}_i|}}$
$\sigma \leftarrow \frac{1}{\sqrt{2\rho}}$
$T \leftarrow \max_{k \in [\Delta_0]} \left\{ \frac{1}{\sqrt{k}} + \sigma \cdot \Phi^{-1}\left((1-\delta)^{1/k}\right) \right\}$
for $u \in Supp(H)$ **do**
  $\hat{H}[u] \leftarrow H[u] + \mathcal{N}(0, \frac{1}{2\rho})$
  if $\hat{H}[u] \geq T$ **then**
    $S \leftarrow S \cup \{u\}$
return S

Theorem 13. *Fix any $\rho > 0$, any $\delta \in (0, 1)$, and any $\Delta_0 \in \mathbb{N}$. The Weighted Gaussian algorithm (Algorithm 2) satisfies δ-approximate ρ-zCDP.*

See Appendix A for the proof of Theorem 13.

The Weighted Gaussian algorithm benefits from being highly scalable. In particular, it lends itself well to parallel computation across several computers within a cluster, because the weights on each histogram item can be computed in parallel as well as the noise addition and thresholding steps (see Figure 8). Thus, Algorithm 2 is the standard approach for doing partition selection on data sets that are too large to fit in a single machine's memory. Unfortunately, Weighted Gaussian suffers from poor accuracy compared to the greedy approaches we discuss next.

3.2 Greedy Approaches

One problem with the Weighted Gaussian algorithm is users waste their sensitivity budget on histogram items that are already well above the threshold. Most real-world data has highly skewed item frequencies, but Weighted Gaussian increments all items in a user's set by the same amount.

The two greedy algorithms we discuss next solve this problem by iterating through the users one-by-one and using an *update policy* and the current state of the histogram to decide how to allocate weight across the items in their set. That is, each user's update depends on previous users' updates. For example, in both algorithms, users do not contribute to items in H that have already reached T^*, the threshold T plus some positive buffer; items that have reached the buffered threshold are very likely to be returned after noise is added and thus do not need more weight. These adaptive update rules are carefully chosen so that the overall algorithm has bounded global sensitivity. As we will discuss in later sections, the main downside of these algorithms is their sequential nature. By design, they require iterating over the entire data set, which may be prohibitively slow for industrial data sets.

3.2.1 Policy Gaussian [11] (DPSU). As with the Weighted Gaussian algorithm, the data set is pre-processed by removing duplicate items from each user's set and truncating the user sets to some fixed maximum size Δ_0. Then, the algorithm iterates sequentially over the users and, for each item u in user i's set W_i, the user increments $H[u]$ by a weight that is proportional to $T^* - H[u]$, where T^* is equal to T plus some positive buffer. Essentially, *items that are further from the buffered threshold get more weight added to them, and those that have already reached the buffered threshold get none.* The weight that a user adds to each item is normalized so a single user's update to H has an ℓ_2-norm of at most 1.We refer to this algorithm as DPSU.

Gopi et al. prove that the global ℓ_2-sensitivity of the entire Policy Gaussian algorithm is bounded by 1, so applying the high-level algorithm (Algorithm 1) with appropriately-scaled Gaussian noise to each item and thresholding are sufficient for satisfying differential privacy.

3.2.2 Greedy updates Without sampling (GW) [6]. Carvalho et al. observe that, rather than removing duplicate items in a user's set, one can use this frequency information to decide where to allocate sensitivity budget. GW iterates over the users, and each user computes u^*: the most frequent item in their list such that $H[u^*]$ is below the buffered threshold T^*. Then, the user increments $H[u^*]$ by $\min(1, T^* - H[u^*], budget)$ where *budget* is the user's remaining ℓ_1 budget. This process is repeated until the user's initial budget of 1 is consumed or the user has no items left that are below T^*[1]. Carvalho et al. prove that their GW algorithm for building a weighted histogram has a global ℓ_1-sensitivity bound of 1, so running the high-level algorithm (Algorithm 1) with Laplace noise and thresholding is sufficient for satisfying differential privacy.

One notable feature of their algorithm is that it can use item frequency information from a public data set to increase the accuracy of the frequency estimates. We do not use this feature when comparing it to other algorithms, since our goal is to create a general-purpose algorithm that could be incorporated into a privacy framework without requiring a data analyst to input a public data set.

4 DP-SIPS

In this section we present DP-SIPS: Differentially Private Scalable, Iterative Partition Selection, detailed in Algorithm 3. The basic structure is quite simple: it runs Weighted Gaussian on the data set multiple times with increasing privacy budget (and corresponding decreasing thresholds); on each iteration, the partitions returned by Weighted Gaussian are removed from each of the users' sets.

Because Weighted Gaussian updates the histogram uniformly across a user's items, when returned partitions are removed from the user's set, they can allocate more weight to their remaining items (see Figure 1). The first iteration returns the very popular items using only a tiny fraction of the overall privacy budget, and subsequent iterations yield less frequent items. The action of the algorithm is twofold: *in each iteration, the threshold is lowered at the same time as users allocate more weight to each of the items that remain in their sets* (since the user sets get smaller when previously-returned items are removed).

Furthermore, the user sets are re-truncated on each iteration after the previously returned partitions are removed. For data sets that are both skewed in the item frequencies and in the sizes of users' sets, this allows additional items to be included on each iteration.

Algorithm 3 DP-SIPS: Scalable, Iterative Partition Selection

Input: Data set $x = (W_1, \dots, W_N)$
Maximum per-user contribution Δ_0
Privacy parameters ρ, δ
Number of iterations I
Privacy budget allocation factor $r > 0$
Output: Subset $S \subseteq \cup_i W_i$

$S \leftarrow \{\}$
for $i = 0, \dots, I-1$ **do**
  if $r = 1$ **then**
    $(\rho_i, \delta_i) \leftarrow \left(\frac{\rho}{I}, \frac{\delta}{I}\right)$
  else
    $(\rho_i, \delta_i) \leftarrow \left(\rho \cdot r^{I-i-1} \cdot \frac{1-r}{1-r^I}, \quad \delta \cdot r^{I-i-1} \cdot \frac{1-r}{1-r^I}\right)$
  $P_i \leftarrow \texttt{Weighted\_Gauss}\,(x = (W_1, \dots, W_N), \rho_i, \delta_i, \Delta_0)$
  for $j \in N$ **do**
    $W_j \leftarrow W_j \setminus P_i$ ▷ Remove already-found partitions
  $S \leftarrow S \cup P_i$
return S

Theorem 14. *For any $\rho > 0$ and any $\delta \in (0, 1]$, Algorithm 3 satisfies δ-approximate ρ-zCDP.*

Proof. By Theorem 13, each call to `Weighted_Gauss` satisfies δ_i-approximate ρ_i-zCDP. Applying composition and postprocessing (Lemma 6), Algorithm 3 satisfies $\left(\sum_{i=0}^{I-1} \delta_i\right)$-approximate $\left(\sum_{i=0}^{I-1} \rho_i\right)$-zCDP.

If $r = 1$, then clearly $\sum_{i=0}^{I-1} \rho/I = \rho$ and similarly $\sum_{i=0}^{I-1} \delta/I = \delta$.

Now, for $r \neq 1$ we will solve for these summations using the closed-form formula for geometric sums.

$$\sum_{i=0}^{I-1} \rho_i = \sum_{i=0}^{I-1} \rho \cdot r^{I-i-1} \cdot \frac{1-r}{1-r^I} = \rho \cdot \frac{1-r}{1-r^I} \cdot \sum_{i=0}^{I-1} r^{I-i-1}$$
$$= \rho \cdot \frac{1-r}{1-r^I} \cdot \sum_{j=0}^{I-1} r^j = \rho \cdot \frac{1-r}{1-r^I} \cdot \frac{1-r^I}{1-r} = \rho.$$

An identical calculation holds for $\sum_{i=0}^{I-1} \delta_i$. Thus, Algorithm 3 satisfies δ-approximate ρ-zCDP. □

A key advantage of DP-SIPS is that it is massively parallelizable: within each iteration, the mechanism performs a simple groupby-count operation, which can be distributed among multiple cores or a cluster of multiple machines. DP-SIPS requires multiple iterations, so it takes longer to run than the naive algorithm. However, only

[1]As the algorithm is presented in [6], it does not always terminate. In particular, they do not consider the case when a user has budget left but all of its items have reached T^*, but adding this condition luckily does not affect the privacy proof.

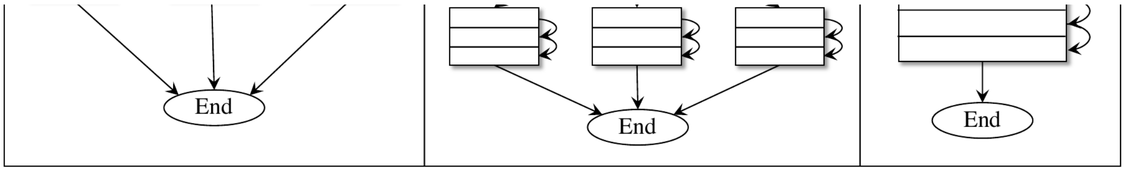

Figure 1: Depiction of Weighted Gaussian noisy histogram (left) compared to intermediate SIPS noisy histograms (right three diagrams) on a skewed data set. Solid blue bars represent partitions that will be returned, and yellow outlines represent the weight of each item on the previous iteration. Although the threshold for Weighted Gaussian is lower than each of the SIPS thresholds, SIPS benefits from less-frequent items getting increased weight as returned partitions are removed from user sets.

a few iterations are required to achieve good accuracy (see Section 5.2.3), so its runtime is still reasonable even on large datasets. A visual explanation of the scalability property of DP-SIPS in comparison with other approaches can be found in Figure 8, and an experimental performance and scalability comparison can be found in Section 5.3.

Relation to Algorithm 7 in [13]. Private Product-Distribution Estimator in [13] privately estimates a product of d Bernoulli distributions, and has some structural similarities to DP-SIPS: it also uses weighted Gaussian iteratively with decreasing thresholds, removing items above the threshold in each round to estimate the frequencies of each coordinate. Aside from the problem setting (bounded vs. unbounded domain) and goal (distribution estimation vs. partition selection), a major difference is that [13] partitions the dataset horizontally into $\log(d/2)$ groups, and uses different subsets of user records on each round. This partitioning step is key to their privacy and accuracy arguments and leads to $\log(d/2)$ rounds rather than the constant number of rounds that DP-SIPS uses.

4.1 Scalability Analysis in the MapReduce Framework

We analyze DP-SIPS in the MapReduce model of distributed computing to better understand its scaling behavior on a cluster. The model proceeds in rounds that are synchronized across worker nodes running in parallel. The data records begin arbitrarily partitioned among the worker nodes. Each round begins with a *map* phase in which the worker node applies a map to each record in its partition. Next, the records are *shuffled* among the workers so that records with the same key are located on the same worker node. Lastly, each worker processes the records in the *reduce* phase. The three phases (map, shuffle, and reduce) constitute one round in the MapReduce framework. Typically, the shuffle phase is the most time intensive since large amounts of data must move among the worker nodes, so the overall number of rounds usually reflects an algorithm's efficiency on a real system.

DP-SIPS proceeds in the following steps (see Figure 2):

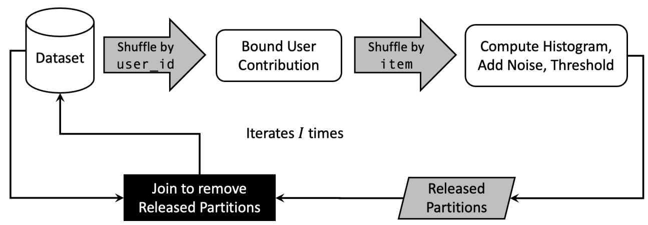

Figure 2: MapReduce Diagram for DP-SIPS. Each iteration within SIPS involves two rounds of MapReduce.

(1) Shuffle records by `user_id`
(2) Bound the number of contributions per user
(3) Shuffle records by `item`. Call this dataset D.
(4) Count the number of items, add noise, and store the set of items S above the threshold
(5) Do a join to remove items in S from D. The records now consist of $D \setminus S$. If this is the last iteration, this step can be skipped.
(6) Return to Step 1 for I iterations in total.

All but the last iteration of DP-SIPS involves three MapReduce rounds: first shuffling records by user to truncate; then shuffling by item to count, add noise, and threshold; lastly, doing a join with the original data to remove items that were released. In the last iteration, the join is not necessary. So, in total the algorithm does $3(I-1)+2$ rounds in the MapReduce framework, where I is the number of iterations of DP-SIPS. In our experiments we set $I = 3$, so DP-SIPS performs 8 MapReduce rounds. Weighted Gaussian only uses 2 MapReduce rounds, since $I = 1$. The greedy algorithms cannot be parallelized–specifically the greedy methods for histogram computation–so each record must be loaded into the one worker node to make its contribution to the histogram. This requires $O(N)$ rounds of MapReduce, where N is the number of users. Indeed, our experimental results in Section 5.3 (Figures 10 and 11) show that the runtime of DP-SIPS decreases with the number of cores while the greedy algorithms stay constant.

Data set	Users	Observations	Vocabulary Size
Reddit	223,388	373,983	155,701
Twitter	702,682	2,811,774	1,300,123
Finance	1,400,465	1,400,465	267,256
IMDb	49,999	49,999	194,532
Wikipedia	245,103	245,103	631,866
Amazon	4,000,000	4,000,000	4,250,427
Synthetic_80M	80,000,000	4,643,596,660	741,129,124

Table 1: Number of users, number of observations, and true vocabulary size for each data set we consider.

5 EXPERIMENTAL RESULTS

We use data sets of several different sizes from varied text domains to validate the empirical performance of the DP-SIPS algorithm, and we find that in general DP-SIPS has comparable accuracy to DPSU and GW while scaling to large data sets that DPSU and GW timeout on. As with prior works, we focus on the problem of vocabulary discovery under the constraint of user-level privacy.

We begin by describing the data sets in Section 5.1 and then, in Section 5.2 we discuss how the empirical accuracy of DP-SIPS compares to that of existing algorithms, and in Section 5.3 we discuss the results from our scalability experiments.

5.1 Data sets

We use six publicly-available data sets and one synthetically-generated data set to study the accuracy of DP-SIPS against existing partition selection algorithms, and we use the largest two of the seven for scalability experiments on Amazon Elastic Map Reduce clusters. `Reddit` [18] is a data set of text posts collected from r/AskReddit which appeared as a benchmark in [6, 11]. We also use four data sets that appeared in [6]: `Twitter` [19], comprising customer support tweets to and from large corporations; `Finance` [3], financial headlines for stocks; `IMDb` [17], a set of movie reviews scraped from IMDb; `Wikipedia` [20], a set of Wikipedia abstracts (where we treat each abstract as a separate user set).

For the scalability experiments, we use `Amazon` [12], a publicly-available text data set from Kaggle of 4 million Amazon product reviews, and a synthetically generated data set `Synthetic_80M` with 80 million users and 4.6 billion observations. The synthetic data set, `Synthetic_80M`, is generated as follows. First, each user draws the number of items in their set from a Pareto distribution of scale 10 and shape 1.16; this captures the common "80-20 principle" observed on many empirical datasets that states that 80% of outcomes are due to 20% of causes. Then, each item in each user's set is generating by sampling from a zeta distribution of parameter 1.1; this captures Zip's law, which states that in many types of data (including words in natural languages), the rank-frequency distribution follows an inverse relation.

Table 1 lists the number of users, number of observations, and the vocabulary size of each data set. Using the same methods as [6, 11], we preprocess all non-synthetic data sets using tokenization with `nltk.word_tokenize`, removing URLs and symbols, and lower casing all words.

5.2 Accuracy results

Because the goal of partition selection is to privately output as many partitions as possible, we measure accuracy as the *number of partitions released.* To date, there are no analytical accuracy guarantees for any prior partition selection algorithms (in the setting where users contribute multiple items), so we must use experimental validation to understand the accuracy of each algorithm.

We test the accuracy of the four algorithms: Weighted Gaussian (Algorithm 2), SIPS (our Algorithm 3), Policy Gaussian (DPSU) from [11], and Greedy updates Without sampling (GW) from [6].

Table 2 shows the following accuracy trends for SIPS on the datasets described in Section 5.1:

- SIPS only performs slightly worse than both GW and DPSU on one dataset (`Finance`); in general, SIPS' performance is on par with DPSU's.
- The accuracy of SIPS is consistently approximately double that of Weighted Gaussian.

In addition, Figures 3, 4, and 5 demonstrate that the relative accuracy of each algorithm on a given data set remains consistent across different choices of privacy parameters.

5.2.1 Accuracy Inconsistencies. While GW tends to perform well, its accuracy on the `IMDb` and `Wikipedia` data sets is below even that of Weighted Gaussian (see Figure 5). To further investigate this phenomenon, we modify these two data sets in addition to `Finance` to remove repeated items within each user's set (see the data sets marked as "deduplicated" in Table 2). Note also that the accuracy of the other three algorithms is unaffected by deduplication since they all perform this preprocessing step to the data sets.

Without any item frequency information, the algorithm adds all of its weight to a randomly-selected item, so one would expect the performance on all three data sets to be worse than Weighted Gaussian. To the contrary, GW's accuracy on the deduplicated `Finance` data set is still significantly higher than the others, and GW's accuracy on the deduplicated `IMDb` and `Wikipedia` is again worse than Weighted Gaussian. Figure 5 shows that GW's poor relative accuracy on `IMDb` worsens at higher levels of epsilon.

We believe that GW's inconsistent accuracy depends on the ratio between vocabulary size and number of observations: GW performs very well on datasets where this ratio is small (`Finance`: 0.19), and very poorly on datasets with large relative vocabulary (`IMDb`: 3.8). Other factors, such as the short length of `Finance` headlines compared to `IMDb` reviews and `Wikipedia` abstracts, might also play a role. We did not attempt to fully resolve this discrepancy in GW's accuracy; however, we present these results as a caution when selecting a partition selection algorithm.

5.2.2 Comparing zCDP to DP. We implement our algorithm to satisfy approximate zCDP to take advantage of its simpler composition properties over standard DP. In [11], the Policy Gaussian algorithm satisfies (ε, δ)-DP, but the threshold and Gaussian noise can easily be recalibrated to satisfy approximate zCDP instead. We do so in this work to facilitate the comparison to our algorithm. Unfortunately, the GW algorithm from [6] has a bounded ℓ_1-sensitivity and uses Laplace noise, so it is not easily converted to satisfy approximate zCDP without a significant loss in accuracy. Instead, we use Corollary 9 to choose appropriate (ε, δ) parameters for the given ρ

Data set	Wt. Gauss	**SIPS**	DPSU	GW
Reddit	6,160	11,392	11,186	11,984
Twitter	12,632	23,649	23,576	27,184
Finance	17,350	27,559	29,005	37,503
IMDb	3,728	7,759	5,845	3,133
Wikipedia	11,340	21,037	18,129	11,251
Amazon	67,522	144,805	143,997	185,563
Synthetic_80M	711,601	1,137,467	-	-
Finance (deduplicated)	-	-	-	37,563
IMDb (deduplicated)	-	-	-	3,005
Wikipedia (deduplicated)	-	-	-	9,802

Table 2: Number of partitions returned by Wt. Gauss (Algorithm 2), SIPS (Algorithm 3), DPSU (Policy Gaussian from [11]), and GW (from [6]) on six data sets. Additionally, we run GW on three data sets where duplicates within user lists have been removed. Note that the other three algorithms already deduplicate user sets. For SIPS we use 3 iterations and $r = 1/3$. For Wt. Gaussian, SIPS, and DPSU, the privacy budget is set to $\rho = 0.1, \delta = 10^{-5}$, and the user contributions are truncated to $\Delta_0 = 100$. For GW, $\varepsilon = 1.7$ and $\delta = 8.1142 \times 10^{-5}$, which implies 10^{-5}-approximate 0.1-zCDP. On Synthetic_80M, DPSU and GW ran out of memory before completing the computation.

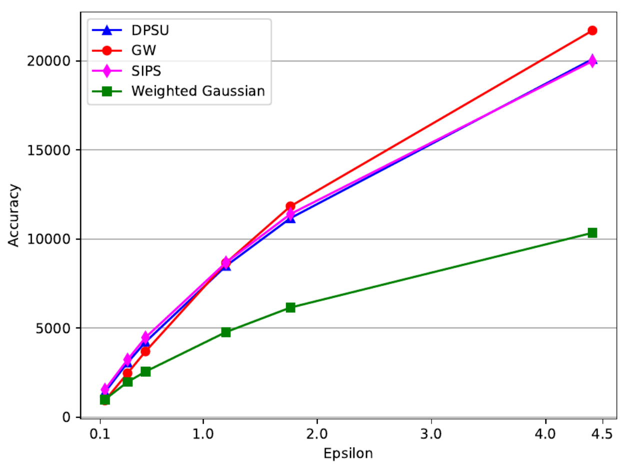

Figure 3: Accuracy on Reddit as a function of varying epsilon (and associated ρ), for $\delta = 10^{-5}$.

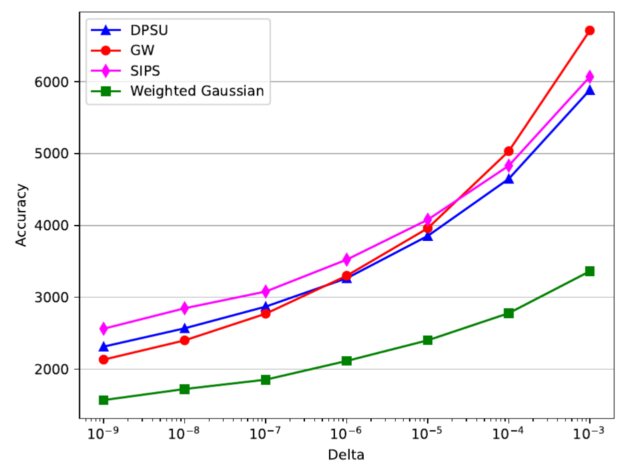

Figure 4: Accuracy on Reddit as a function of varying delta for $\varepsilon = 1.7$.

and δ_{CDP}. The tables in Appendix B give the conversions derived from Corollary 9 used for our experiments.

The conversion from approximate zCDP to approximate DP is not exactly tight (meaning the given (ε, δ) may be higher than the true privacy guarantee given by the approximate zCDP parameters). Because of the looseness of the conversion, the GW algorithm's accuracy may be slightly inflated when compared to the other algorithms.

Doing the privacy budget accounting using approximate zCDP is not exactly tight either; instead, one could use numerical methods to compute the total privacy budget spent [16]. At the time of writing, we could not find a working implementation of such methods that could support the privacy property of Weighted Gaussian. Doing so could slightly improve the privacy analysis of DP-SIPS, leading to more favorable comparisons with other approaches. Furthermore, using approximate zCDP has the advantage of being easier to integrate with existing differential privacy software [2, 15].

Doing the privacy budget accounting with zCDP is not exactly tight either: numerical methods using

5.2.3 Selecting hyperparameters. Our algorithm has several hyperparameters (aside from the privacy parameters ρ and δ) that need to be set by the data analyst: Δ_0 the maximum number of items per user, r the ratio between the privacy budget for iteration $i+1$ and i, and I the number of iterations in the algorithm. One option is to divide the privacy budget and try several hyperparameter settings and select the setting with the highest accuracy; however, this wastes a lot of privacy budget. We find that the accuracy of DP-SIPS is largely invariant to reasonable settings of the hyperparameters, and we provide general rules of thumb for selecting them.

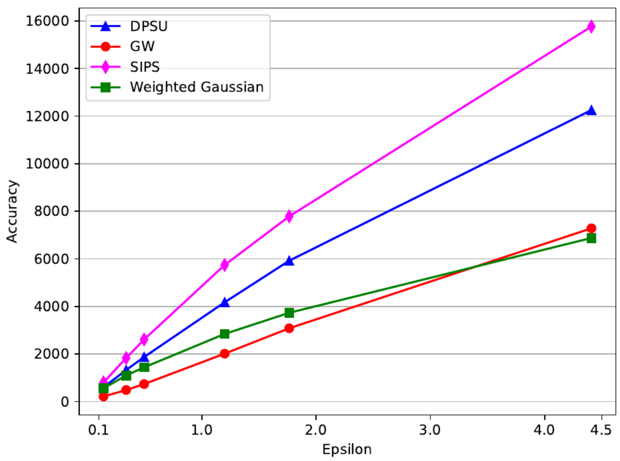

Figure 5: Accuracy on `IMDb` as a function of varying epsilon for fixed $\delta = 10^{-5}$.

Iterations I	Partitions Returned
1	5,464
2	10,041
3	11,126
4	11,182
5	11,541
6	11,061
8	11,585
10	11,637

Table 3: SIPS's accuracy as a function of the number of iterations on `Reddit` with parameters $\rho = 0.1, \delta = 10^{-5}$, $r = 1/3$, and $\Delta_0 = 50$.

Figures 6 and 7 display the accuracy of our algorithm on five data sets with varying r and Δ_0, respectively. For the given setting of δ and ρ, the accuracy of DP-SIPS is largely unaffected by different choices of r and Δ_0 across 5 data sets, and Table 3 shows that the accuracy of DP-SIPS only slightly increases with the number of iterations I on the `Reddit` data set (for the given settings of the other parameters). Furthermore, Figure 6 (privacy ratio $r = 1$) suggests that evenly splitting the budget among the rounds yields lower accuracy than geometrically increasing the budget at each round.

The results suggest that DP-SIPS has good accuracy for a large range of hyperparameters (aside from the privacy parameters). We recommend using the following settings: $r \in [0.2, 0.4]$, $I \geq 3$, and Δ_0 set to an overestimate of the true maximum number of per-user contributions. If the true maximum number of contributions per user is public, Δ_0 should be set to that value.

5.3 Scalability results

We benchmark the algorithms in several ways, and we find that DP-SIPS and Weighted Gaussian scale well with large data sets, while DPSU and GW do not.

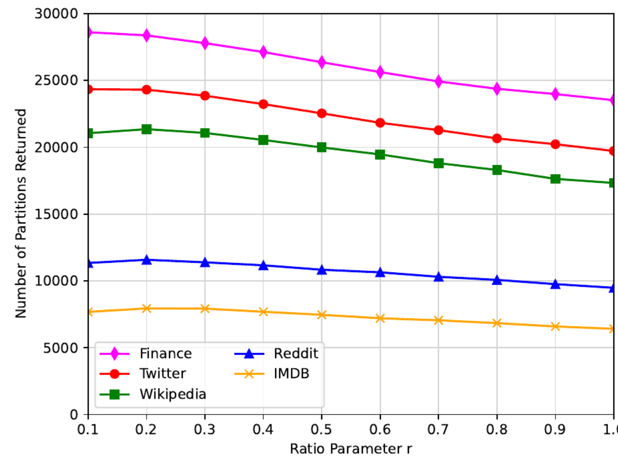

Figure 6: Number of partitions returned by SIPS versus the choice of the privacy ratio parameter, r. All other hyperparameters are identical to those listed in Table 2. For all 5 data sets, the accuracy is maximized for $r \in [0.1, 0.4]$.

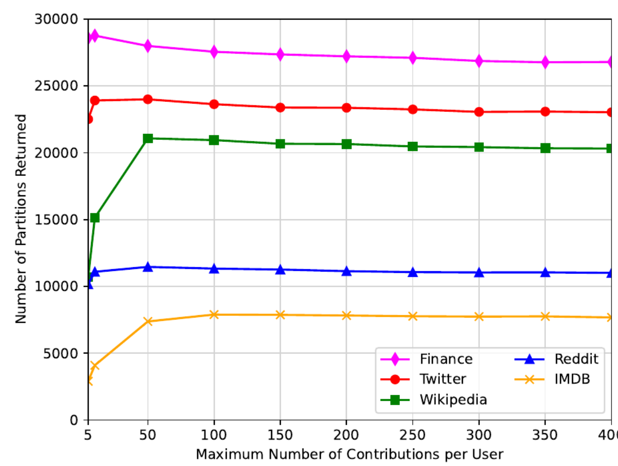

Figure 7: Number of partitions returned by SIPS for varying per-user maximum contribution bounds, Δ_0. All other hyperparameters are identical to those listed in Table 2. On all 5 data sets, increasing Δ_0 past 100 has almost no affect on accuracy, with maximum accuracy for $\Delta_0 \in [5, 150]$. We note that `Finance` is unusual as each financial headline is quite short.

For the scalability experiments, we implement all of the algorithms in PySpark to take advantage of parallelism within the algorithms. We then run the algorithms on Amazon Elastic Map Reduce (EMR) clusters, and we tailor the PySpark session settings to the number of cores and memory allocation for both driver and executor nodes to reflect the available resources of the machines.

To benchmark the algorithms, we measure the amount of time required to run the algorithm (after the dataset has already been read

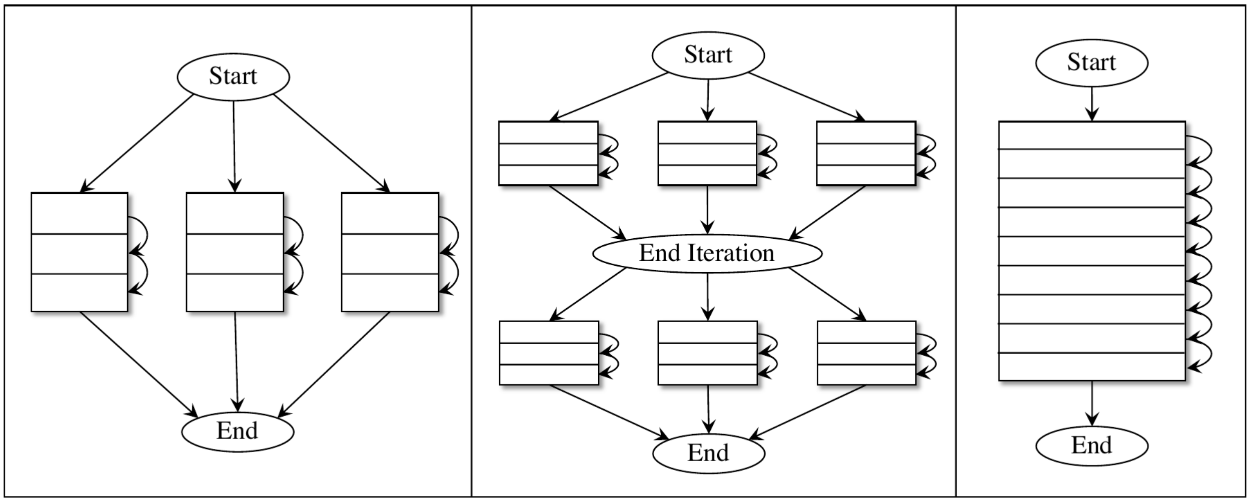

Figure 8: Parallelism diagram for Weighted Gaussian (left), SIPS (middle), and greedy algorithms (right). Arrows represent computational steps while boxes represent the data set. The steps of Weighted Gaussian can run in parallel on parts of the data set while greedy algorithms must run sequentially over the data set. Each iteration of SIPS can run in parallel, but the iterations must be done sequentially.

in to PySpark) and return the number of partitions that were discovered, in order to ensure PySpark's lazy evaluation executed the algorithm during the timing phase. See Appendix C for information about the machine specifications.

For our first scalability experiment, we run the algorithms on subsets of `Synthetic_80M`, a synthetic data set with 80 million users and 4.6 billion items in total, and benchmark the algorithms as the number of users increases. Figure 9 shows the results of this experiment. DP-SIPS and Weighted Gaussian scale well to large data sets while DPSU and GW timeout or run out of memory on subsets with just 2 million and 10 million users, respectively. We ran this experiment on an EMR cluster with 8 nodes, each of size `m5a.8xlarge` (each node has 32 processor cores and 128 GB of memory).

To further investigate the scaling behaviors of the algorithms, we ran more scalability experiments on a smaller data set, `Amazon`, which consists of product reviews from 4 million users. Figure 10 shows the results from the first experiment, in which we measure the runtimes of the algorithms on clusters with a single master node and varying numbers of core nodes. All nodes are of type m5a.2xlarge, which has 8 processor cores and 32 GB of memory. Figure 10 illustrates the following runtime trends:

- The runtimes of Weighted Gaussian and SIPS both decrease as the number of core nodes increases.
- The runtimes of DPSU and GW remain approximately the same even as the number of core nodes increases.

These results confirm the intuition that Weighted Gaussian and SIPS are both parallelizable and thus scale well with increased cluster sizes, even on large data sets. Additionally, it confirms our observation that since DPSU and GW need to iterate sequentially

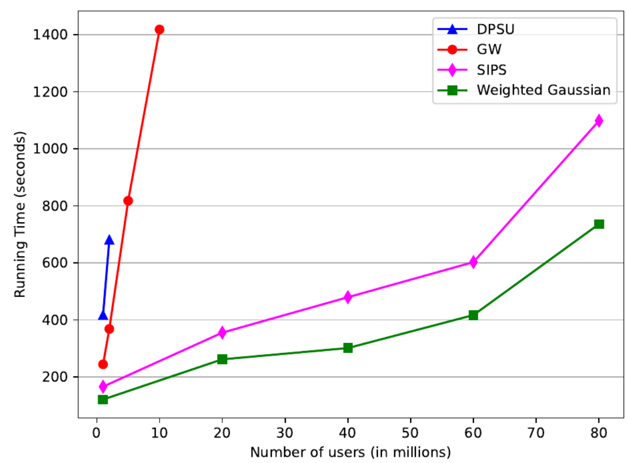

Figure 9: Algorithm runtimes on subsets of `Synthetic_80M` of given sizes on a cluster of 8 `m5a.8xlarge` nodes. The algorithms were run with the same hyperparameters as those listed in Table 2. DPSU and GW timed out after subsets with 2 million and 10 million users, respectively.

over each user, increasing the cluster size does little to improve their running times.

We run an additional scalability experiment on `Amazon`: instead of increasing the number of core nodes, we increase the sizes of the core nodes. We use a single master node and two core nodes, and increase the sizes of all three nodes. Table 11 shows similar trends as the previous experiment: Weighted Gauss and SIPS scale with

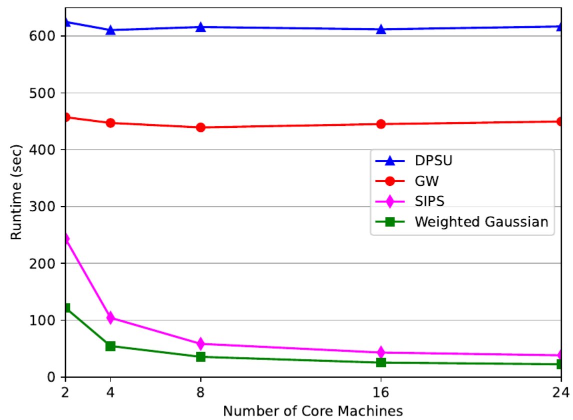

Figure 10: Algorithm runtimes on Amazon dataset with increasing number of m5a.2xlarge cores in the cluster. The algorithms were run with the same hyperparameters as those listed in Table 2.

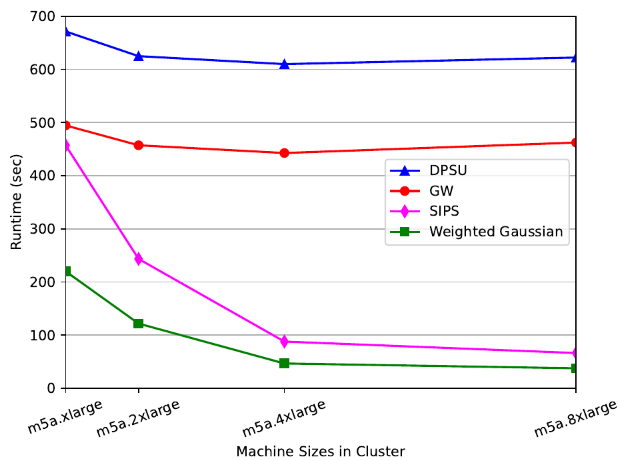

Figure 11: Algorithm runtimes on Amazon dataset with increasing sizes of nodes in a cluster with 1 master node and 2 core nodes. The algorithms were run with the same hyperparameters as those listed in Table 2.

increased node sizes while DPSU and GW do not. So, even when the sizes of the machines increase, this makes little difference in the runtimes of DPSU and SIPS.

Comparing Figures 10 and 11, we see that for DP-SIPS, the communication overhead between machines is small. Specifically we can compare SIPS's performance on 8 cores in Figure 10, a cluster with 1 master node and 8 core nodes each of type m5a.2xlarge (8 CPU cores, 32 GB memory) to m5a.8xlarge (32 CPU cores, 128 GB memory) in Figure 11, a cluster with 1 master node and 2 core nodes. In both, the core nodes have in total $8 \cdot 8 = 2 \cdot 32$ CPU cores and $8 \cdot 32 = 2 \cdot 128$ GB of memory among them. The runtimes of SIPS in the first case is 58.41 seconds while in the second case, it is 66.28 seconds, which suggests that the communication overhead is small. In fact, SIPS runs faster on a cluster of 9 smaller machines than a cluster of 3 larger machines. This discrepancy could be the result of any number of factors in the machine specifications.

Our takeaway is clear: the scalability experiments confirm that DPSU and GW are not suitable for massive data sets and the prohibitively poor run times of DPSU and GW on industrial-scale data sets cannot be overcome by increasing the computational power of the cluster (either in number of machines or even machine size) because the algorithms are not parallelizable on a fundamental level. For larger data sets, SIPS and Weighted Gaussian are the only feasible algorithms and SIPS consistently has improved accuracy over Weighted Gaussian.

6 DISCUSSION

Differentially private partition selection (or set union) is a fundamental problem for many private data analysis tasks. Prior approaches to this problem either suffer from poor accuracy or are prohibitively slow on large data sets because they are designed to sequentially iterate over the users. We present a simple algorithm for differentially private partition selection that achieves accuracy that is comparable to DPSU and runtimes that scale well with increased computational resources.

6.1 Unsuccessful Attempts

One natural question we explored is: can we boost the accuracy of DPSU using our method of iterating several times and removing partitions from the data set that were previously returned? We found that empirically this did not boost the accuracy on the data sets that we tested. Intuitively, this makes sense because the per-user Gaussian update policy in DPSU prevents users from adding weight to items that have already reached the buffered threshold. Such items will very likely be released after the noise addition and thresholding steps. The DPSU approach achieves the same goal as iterating, albeit with more precision due to the greedy nature of the Policy Gaussian per-user updates.

6.2 Future work

This work raises several natural questions. From a practical perspective, one may wonder whether there is a scalable algorithm with higher accuracy than DP-SIPS. Another interesting line of inquiry could consider theoretical guarantees on the number of partitions released, possibly under some distributional assumptions on the data. We believe these lines of inquiry would give important insights into a fundamental problem in private data analysis.

ACKNOWLEDGMENTS

We are thankful to the anonymous reviewers for the helpful feedback. This work was supported by Tumult Labs.

REFERENCES

[1] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. Plume: Differential privacy at scale. *arXiv preprint arXiv:2201.11603*, 2022.

[2] Skye Berghel, Philip Bohannon, Damien Desfontaines, Charles Estes, Sam Haney, Luke Hartman, Michael Hay, Ashwin Machanavajjhala, Tom Magerlein, Gerome Miklau, Amritha Pai, William Sexton, and Ruchit Shrestha. Tumult Analytics: a robust, easy-to-use, scalable, and expressive framework for differential privacy. *arXiv preprint arXiv:2212.04133*, December 2022.

[3] Bot_Developer. Daily financial news for 6000+ stocks. https://www.kaggle.com/datasets/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests.

[4] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

[5] Clément L Canonne, Gautam Kamath, and Thomas Steinke. The Discrete Gaussian for Differential Privacy. In *Advances in Neural Information Processing Systems*, volume 33, pages 15676–15688. Curran Associates, Inc., 2020.

[6] Ricardo Silva Carvalho, Ke Wang, and Lovedeep Singh Gondara. Incorporating item frequency for differentially private set union. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9504–9511, 2022.

[7] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 486–503. Springer, 2006.

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[9] Google. Privacy on Beam. https://github.com/google/differential-privacy/tree/main/privacy-on-beam, November 2022.

[10] Google and OpenMined. PipelineDP. https://pipelinedp.io, November 2022.

[11] Sivakanth Gopi, Pankaj Gulhane, Janardhan Kulkarni, Judy Hanwen Shen, Milad Shokouhi, and Sergey Yekhanin. Differentially private set union. In *International Conference on Machine Learning*, pages 3627–3636. PMLR, 2020.

[12] Kritanjali Jain. Amazon reviews. https://www.kaggle.com/datasets/kritanjalijain/amazon-reviews.

[13] Gautam Kamath, Jerry Li, Vikrant Singhal, and Jonathan Ullman. Privately learning high-dimensional distributions. In *Conference on Learning Theory*, pages 1853–1902. PMLR, 2019.

[14] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. Releasing search queries and clicks privately. In *Proceedings of the 18th international conference on World wide web*, pages 171–180, 2009.

[15] Tumult Labs. Tumult Analytics. https://tmlt.dev, December 2022.

[16] Sebastian Meiser and Esfandiar Mohammadi. Tight on budget? tight bounds for r-fold approximate differential privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 247–264, 2018.

[17] Lakshmipathi N. Imdb dataset of 50k movie reviews. https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

[18] Judy Hanwen Shen. Ask reddit. https://github.com/heyyjudes/differentially-private-set-union/tree/ea7b39285dace35cc9e9029692802759f3e1c8e8/data.

[19] Thought Vector and Stuart Axelbrooke. Customer support on twitter. https://www.kaggle.com/datasets/thoughtvector/customer-support-on-twitter.

[20] Mark Wijkhuizen. Simple/normal wikipedia abstracts v1. https://www.kaggle.com/datasets/markwijkhuizen/simplenormal-wikipedia-abstracts-v1.

[21] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private SQL with bounded user contribution. *Proceedings on Privacy Enhancing Technologies*, 2:230–250, 2020.

A PROOF OF THEOREM 13

Proof of Theorem 13. Since the algorithm is simply computing a stable histogram, the proof follows from standard arguments.

Let us denote Algorithm 2 by $\mathcal{M}$. Fix any $\rho > 0$, any $\delta \in (0, 1)$, and any $\Delta_0 \in \mathbb{N}$. Fix two neighboring data sets $x = (W_1, \dots, W_N)$ and x', where x' contains one extra user set W^*. Let E be the event that $\mathcal{M}(x') \subseteq \cup_{i\in[N]}\overline{W_i}$; that is, the partitions returned for data set x' are in the support of $\mathcal{M}(x)$. We will first argue that, conditioned on event E, the output distributions of $\mathcal{M}(x)$ and $\mathcal{M}(x')$ satisfy pure zCDP (Definition 4). Then we will argue that, because of the thresholding step, event E occurs with probability at least $1 - \delta$.

If we condition both $\mathcal{M}(x)$ and $\mathcal{M}(x')$ on event E, then by the Gaussian mechanism (Definition 12) and post-processing (Lemma 6), Algorithm 2 satisfies pure zCDP (Definition 4).

Now, it remains to show that $\Pr[E] \geq 1 - \delta$. First, note that each user contributes to most Δ_0 items in the histogram, and further note that event E^c occurs when at least one item from W^* that is not in x is released. Let W' denote the set of items in W^* that do not appear in x. Then,

$$\begin{aligned}
\Pr[E] &= \Pr[\forall u \in W', u \notin \mathcal{M}(x)] \\
&= \Pr[\cap_{u\in W'} \hat{H}[u] \leq T] \\
&= \Pr[\cap_{u\in W'} H[u] + Z_u \leq T] \quad \text{For } Z_u \sim \mathcal{N}(0, 1/2\rho) \\
&= \prod_{u\in W'} \Pr[H[u] + Z_u \leq T] \quad \text{By independence of } Z_u\text{'s} \\
&= \prod_{u\in W'} \Pr\left[\frac{1}{\sqrt{|W'|}} + Z_u \leq T\right] \\
&= \left(\Pr\left[\frac{1}{\sqrt{|W'|}} + Z_u \leq T\right]\right)^{|W'|} \quad \text{Since } Z_u\text{'s are i.i.d.} \\
&\geq \min_{k\in[\Delta_0]} \left\{\left(\Pr\left[\frac{1}{\sqrt{k}} + Z_u \leq T\right]\right)^k\right\} \quad \text{Since } |W'| \leq \Delta_0 \\
&\geq 1 - \delta \quad \text{By definition of } T.
\end{aligned}$$

Therefore, $\mathcal{M}$ is δ-approximate ρ-zCDP. □

B PRIVACY PARAMETER CONVERSIONS

Tables 4 and 5 give the conversions between zCDP and DP that we use for the experiments, as well as the values of α that are used in Corollary 9 to do the conversions.

ρ	δ_{CDP}	ε	δ_{DP}	α
0.001	1×10^{-5}	0.14	5.00×10^{-5}	77.033
0.005	1×10^{-5}	0.338	5.08×10^{-5}	37.037
0.01	1×10^{-5}	0.495	4.99×10^{-5}	27.128
0.05	1×10^{-5}	1.2	4.99×10^{-5}	13.283
0.1	1×10^{-5}	1.765	4.96×10^{-5}	9.86
0.5	1×10^{-5}	4.41	4.90×10^{-5}	5.127

Table 4: Values of $\rho, \delta_{CDP}, \varepsilon$, and δ_{DP} used for experiments with fixed δ_{DP} and varying ε.

ρ	δ_{CDP}	ε	δ_{DP}	α
0.005	1×10^{-9}	0.62	1.04×10^{-9}	64.073
0.0055	1×10^{-8}	0.62	1.02×10^{-8}	58.443
0.006	1×10^{-7}	0.62	1.01×10^{-7}	53.732
0.007	1×10^{-6}	0.62	1.01×10^{-6}	46.334
0.0083	1×10^{-5}	0.62	1.01×10^{-5}	39.398
0.01	1×10^{-4}	0.62	1.01×10^{-4}	33.037
0.013	1×10^{-3}	0.62	1.01×10^{-3}	25.863

Table 5: Values of $\rho, \delta_{CDP}, \varepsilon$, and δ_{DP} used for experiments with fixed ε and varying δ_{DP}.

C AMAZON EMR CLUSTER SPECIFICATIONS

Table 6 shows the number of CPU cores and amount of memory available to each type of machine on Amazon EMR that we use in our experiments.

Machine Size	Cores	Memory (GB)
m5a.xlarge	4	16
m5a.2xlarge	8	32
m5a.4xlarge	16	64
m5a.8xlarge	32	128

Table 6: Number of cores and memory per EMR machine of each size.