Journal of Privacy and Confidentiality
Vol. 12 (2) 2022

TPDP 2021

Submitted Dec 2021
Published Nov 2022

REPRESENTING SPARSE VECTORS WITH DIFFERENTIAL PRIVACY, LOW ERROR, OPTIMAL SPACE, AND FAST ACCESS

MARTIN AUMÜLLER, CHRISTIAN JANOS LEBEDA, AND RASMUS PAGH

IT University of Copenhagen, Denmark
e-mail address: maau@itu.dk

BARC and IT University of Copenhagen, Denmark
e-mail address: chle@itu.dk

BARC and University of Copenhagen, Denmark
e-mail address: pagh@di.ku.dk

Abstract. Representing a sparse histogram, or more generally a sparse vector, is a fundamental task in differential privacy. An ideal solution would require space close to information-theoretical lower bounds, have an error distribution that depends optimally on the desired privacy level, and allow fast random access to entries in the vector. However, existing approaches have only achieved two of these three goals.

In this paper we introduce the Approximate Laplace Projection (ALP) mechanism for approximating k-sparse vectors. This mechanism is shown to simultaneously have information-theoretically optimal space (up to constant factors), fast access to vector entries, and error of the same magnitude as the Laplace mechanism applied to dense vectors. A key new technique is a *unary* representation of small integers, which we show to be robust against "randomized response" noise. This representation is combined with hashing, in the spirit of Bloom filters, to obtain a space-efficient, differentially private representation.

Our theoretical performance bounds are complemented by simulations showing that the constant factors on the main performance parameters are quite small and supporting practicality of the technique.

1. Introduction

One of the fundamental results in differential privacy is that a histogram can be made differentially private by adding noise from the Laplace distribution to each entry of the histogram before it is released [DMNS16]. The expected magnitude of the noise added to each histogram entry is $O(1/\varepsilon)$, where ε is the privacy parameter, and this is known to be optimal [HT10]. In fact, there is a sense in which the Laplace mechanism is optimal [KHP15]. However, some histograms of interest are extremely sparse, and cannot be represented in

Key words and phrases: Algorithms, differential privacy, sparse vectors, histograms.

The conference version of this paper was published as M. Aumüller, C. J. Lebeda, R.Pagh: *Differentially Private Sparse Vectors with Low Error, Optimal Space, and Fast Access.* at ACM CCS 2021, pp. 1223-1236.

JPrivacy Confidentiality
www.journalprivacyconfidentiality.org
DOI:10.29012/jpc.809

explicit form. Consider, for example, a histogram of the number of HTTP requests to various servers. Already the IPv4 address space has over 4 billion addresses, and the number of unique, valid URLs has long exceeded 10^{12}, so it is clearly not feasible to create a histogram with a (noisy) counter for each possible value.

Korolova, Kenthapadi, Mishra, and Ntoulas [KKMN09] showed that it is possible to achieve *approximate* differential privacy with space that depends only on the number of non-zero entries in the histogram. However, for (ε, δ)-differential privacy the upper bound on the expected per-entry error becomes $O(\log(1/\delta)/\varepsilon)$, which is significantly worse than the Laplace mechanism for small δ. Cormode, Procopiuc, Srivastava, and Tran [CPST12] showed how to achieve pure ε-differential privacy with expected per-entry error bounded by $O(\log d/\varepsilon)$, where d is the dimension of the histogram, i.e., the number of entries including zero entries. While both these methods sacrifice accuracy, they are very fast, allowing access to entries of the private histogram in constant time. If access time is not of concern, it is possible to combine small space with small per-entry error, as shown by Balcer and Vadhan [BV19]. They achieve an error distribution that is comparable to the Laplace mechanism (up to constant factors) and space proportional to the sum n of all histogram entries—but the time to access a single entry is $\tilde{O}(n/\varepsilon)$, which is excessive for large datasets.

1.1. **Our results.** Our contribution is a mechanism that achieves optimal error and space (up to constant factors) with only a small increase in access time. The mechanism works for either approximate or pure differential privacy, with the former providing faster access time. Our main results are summarized in Theorem 1.1.

Theorem 1.1 (Informal Version of Theorems 5.10 and 5.11)**.** *Let x be a histogram with d entries each bounded by some value u, in which at most k entries have non-zero values. Given privacy parameters $\varepsilon > 0$ and $\delta \geq 0$, there exists an (ε, δ)-differentially private algorithm to represent x with per-entry error matching the Laplace mechanism up to constant factors and the following properties:*

- *If $\delta = 0$, the representation uses $O(k \log(d + u))$ bits and the access time is $O(\log d)$.*
- *If $\delta > 0$, it uses $O(k \log(d + u) + k \log(1/\delta))$ bits and the access time is $O(\log(1/\delta))$.*

For simplicity, the memory requirement does not include the space needed for storing hash functions. Asymptotically, this additional cost shows up for $k = o(\log d)$ as an additional $O(\log^2 d)$ or $O(\log(1/\delta) \log d)$ term. See the theorem statements in Section 5 for more details.

We present variations on this central result in Sections 6–8. These results provide a tighter error analysis for large values of ε, as well as a way to improve the access time. The improvements in running time sacrifice the property that the per-entry error matches the error of the Laplace mechanism. Table 1 shows an overview of notable properties for each variation.

1.2. **Techniques.** On a high level, we treat "small" and "large" values of the histogram differently. Large values are handled by the thresholding technique developed in [KKMN09, CPST12]. We represent small entries as fixed-length bit strings using a *unary encoding*. From [KKMN09, CPST12], we know that their length is logarithmic in either d (for ε-DP) or $1/\delta$ (for (ε, δ)-DP). Privacy is achieved by perturbing each bit using randomized response [War65]. The value of an entry is estimated by finding the unary encoding that is closest in Hamming distance to the noisy bit string. As it turns out, the unary encoding is

	Section 5	Section 6	Section 7	Section 8
Evaluation time	$O(\log d)$	$O(\log(\log(d)/\varepsilon))$	$O(\log d)$	$O(1)$
Error guarantee	Additive	Multiplicative	Additive	Additive
Input domain	Reals	Reals	Integers	Reals
Strong tail bounds	Yes	No	Yes	No

TABLE 1. Informal overview of our main results for pure differential privacy. Each mechanism also has an approximate differentially private version for which $1/\delta$ replaces d in evaluation time.

redundant enough to allow accurate estimation even when the probability of flipping each bit is a constant bounded away from $1/2$. In order to pack all unary representations into small space, we use hashing to randomize the position of each bit in the unary representation of a given entry. The access time is linear in the length of the bit representation, given constant time evaluation of the hash function. Interestingly, although hash collisions can lead to overestimates, they do not influence the asymptotic error.

We remark here that a direct application of randomized response does not give the desired $O(1/\varepsilon)$ error dependency, but we solve this issue with an initial scaling step that gives ε-differential privacy when combined with randomized response. Though the discussion above has been phrased in terms of histograms, which makes the comparison to earlier work easier, our techniques apply more generally to representing sparse real vectors, with privacy for neighboring datasets with bounded ℓ_1-distance. We also discuss a variant of our mechanism for the special case of histograms. Here it is possible to get $O\left(1/e^{\varepsilon}\right)$ error dependency, which is preferred in the low privacy setting.

1.3. **Overview.** In Section 2 we define differential privacy for vectors, discuss the Laplace mechanism, and provide probabilistic tools necessary for the analysis. In Section 3 we discuss related work on differentially private sparse histograms. In Section 4 we introduce the Approximate Laplace Projection (ALP) mechanism and analyze its theoretical guarantees. In Section 5 we improve space and access time using techniques from earlier work [KKMN09, CPST12]. Section 6 discusses using the ALP mechanism on bit length-encoded coordinates and shows that this improves the running time while incurring a multiplicative error. Section 7 discusses a tighter utility analysis for the special case of histograms. Section 8 discusses a data structure that achieves constant access time with expected error $O(1/\varepsilon)$, but with weak tail bounds. In Section 9 we evaluate the performance of the ALP mechanism based on simulations. In Section 10 we present suggestions for practical applications. We conclude the paper by stating an open problem in Section 11.

2. PRELIMINARIES

Problem Setup. In this work, we consider d-dimensional k-sparse vectors of nonnegative real values. We say that a vector $x \in \mathbb{R}_+^d$ is k-sparse if it contains at most k non-zero entries.

All entries are bounded from above by a value $u \in \mathbb{R}$, i.e., $\max_{i\in[d]} x_i =: \|x\|_\infty \leq u$. Here $[d]$ is the set of integers $\{1, \ldots, d\}$. We consider the problem of constructing an algorithm $\mathcal{M}$ for releasing a differentially private representation of x, i.e., $\tilde{x} := \mathcal{M}(x)$. Note that $\tilde{x}$ does not itself need to be k-sparse. In fact, Balcer and Vadhan [BV19] provided a lower bound for the error of any differentially private mechanism that always outputs a sparse vector. We discuss this further in Section 3.

Utility Measures. We use two measures for the utility of an algorithm $\mathcal{M}$. We define the *per-entry error* as $|x_i - \tilde{x}_i|$ for any $i \in [d]$. We define the *maximum error* as $\max_{i\in[d]} |x_i - \tilde{x}_i| = \|x - \tilde{x}\|_\infty$. We compare the utility of algorithms using the expected per-entry and maximum error and compare the tail probabilities of the per-entry error of our algorithm with the Laplace mechanism introduced below.

Differential Privacy. Differential privacy is a constraint to limit privacy loss, introduced by Dwork, McSherry, Nissim, and Smith [DMNS16]. We use definitions and results as presented by Dwork and Roth [DR14]. Intuitively, a differentially private algorithm ensures that a slight change in the input does not significantly impact the probability of seeing any particular output. We measure the distance between inputs using their ℓ_1-distance. In this work, two vectors are neighbors iff their ℓ_1-distance is at most 1. That is for all neighboring vectors $x, x' \in \mathbb{R}^d_+$ we have $\|x - x'\|_1 := \sum_{i\in[d]} |x_i - x'_i| \leq 1$. We can now define differential privacy for neighboring vectors.

Definition 2.1 (Differential privacy [DR14, Def 2.4])**.** Given $\varepsilon > 0$ and $\delta \geq 0$, a randomized algorithm $\mathcal{M} \colon \mathbb{R}^d_+ \to \mathcal{R}$ is (ε, δ)-differentially private if for all subsets of outputs $S \subseteq \mathcal{R}$ and pairs of k-sparse input vectors $x, x' \in \mathbb{R}^d_+$ such that $\|x - x'\|_1 \leq 1$ it holds that:

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

$\mathcal{M}$ satisfies *approximate differential privacy* when $\delta > 0$ and *pure differential privacy* when $\delta = 0$. In particular, a pure differentially private algorithm satisfies *ε-differential privacy*. The following properties of differential privacy are useful in this paper.

Lemma 2.2 (Post-processing [DR14, Proposition 2.1])**.** *Let $\mathcal{M} \colon \mathbb{R}^d_+ \to \mathcal{R}$ be an (ε, δ)-differentially private algorithm and let $f \colon \mathcal{R} \to \mathcal{R}'$ be any randomized mapping. Then $f \circ \mathcal{M} \colon \mathbb{R}^d_+ \to \mathcal{R}'$ is (ε, δ)-differentially private.*

Lemma 2.3 (Composition [DR14, Theorem 3.16])**.** *Let $\mathcal{M}_1 \colon \mathbb{R}^d_+ \to \mathcal{R}_1$ and $\mathcal{M}_2 \colon \mathbb{R}^d_+ \to \mathcal{R}_2$ be randomized algorithms such that M_1 is $(\varepsilon_1, \delta_1)$-differentially private and $\mathcal{M}_2$ is $(\varepsilon_2, \delta_2)$-differentially private. Then the algorithm $\mathcal{M}$ where $\mathcal{M}(x) = (\mathcal{M}_1(x), \mathcal{M}_2(x))$ is $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-differentially private.*

Throughout this paper, we restrict the output of all algorithms to the interval $[0, u]$. An estimate outside this interval is due to noise, and restricting outputs cannot increase the error. It follows from Lemma 2.2 that restricting the output does not affect privacy. We restrict the output implicitly to simplify presentation.

Probabilistic Tools. *The Laplace Mechanism* introduced by Dwork, McSherry, Nissim, and Smith [DMNS16] satisfies pure differential privacy by adding noise calibrated to the ℓ_1-distance to each entry. For completeness, Algorithm 1 provides a formulation of the Laplace mechanism in the context of releasing an ε-differentially private representation of a sparse vector.

Algorithm 1: The Laplace Mechanism

Parameters: $\varepsilon > 0$.
Input : k-sparse vector $x \in \mathbb{R}_+^d$.
Output : ε-differentially private approximation of x.

(1) Let $\tilde{x}_i = x_i + \eta_i$ for all $i \in [d]$, where $\eta_i \sim \mathrm{Lap}(1/\varepsilon)$ is sampled independently.
(2) Release $\tilde{x}$.

Here $\mathrm{Lap}(1/\varepsilon)$ is the Laplace distribution with scale parameter $1/\varepsilon$. The PDF and CDF of the distribution are presented in Definition 2.4 and the expected error and tail bound of the mechanism are shown in Proposition 2.5. The Laplace mechanism works well for vectors with low dimensionality and serves as a baseline for our work. However, it is impractical or even infeasible in the setting of k-sparse vectors. The output vector is dense, and as such the space requirement scales linearly in the input dimensionality d.

Definition 2.4. The probability density and cumulative distribution functions of the Laplace distribution centered around 0 with scale parameter $1/\varepsilon$ are

$$f(\tau) = \frac{\varepsilon}{2} e^{-|\tau|\varepsilon} \ .$$

$$\Pr[\mathrm{Lap}(1/\varepsilon) \le \tau] = \begin{cases} \frac{1}{2} e^{\tau\varepsilon}, & \text{if } \tau < 0 \\ 1 - \frac{1}{2} e^{-\tau\varepsilon}, & \text{if } \tau \ge 0 \end{cases}$$

Proposition 2.5 (Expected Error and Tail Bound [DR14, Theorem 3.8])**.** *The expected per-entry error and the maximum error of the Laplace mechanism are* $\mathbb{E}[|x_i - \tilde{x}_i|] = O(1/\varepsilon)$ *and* $\mathbb{E}[\|x - \tilde{x}\|_\infty] = O(\log d/\varepsilon)$*, respectively. With probability at least* $1 - \psi$ *we have*

$$|\mathrm{Lap}(1/\varepsilon)| \le \frac{1}{\varepsilon} \ln \frac{1}{\psi} \ .$$

Random rounding (also known as stochastic rounding) is used for rounding a real value probabilistically based on its fractional part. We define random rounding for any real $r \in \mathbb{R}$ as follows:

$$\mathrm{RandRound}\,(r) = \begin{cases} \lceil r \rceil & \text{with probability } r - \lfloor r \rfloor \\ \lfloor r \rfloor & \text{with probability } 1 - (r - \lfloor r \rfloor) \end{cases}$$

Lemma 2.6. *The expected error of random rounding is maximized when* $r - \lfloor r \rfloor = 0.5$*. For any* r *we have*

$$\mathbb{E}[|r - \mathrm{RandRound}\,(r)\,|] \le \frac{1}{2} \ .$$

Randomized response was first introduced by Warner [War65]. The purpose of the mechanism is to achieve plausible deniability by changing one's answer to some question

Algorithm	Space (bits)	Access time	Per-entry error	Maximum error
Dwork et al. [DMNS16]	$O(d \log(u))$	$O(1)$	$O\left(\frac{1}{\varepsilon}\right)$	$O(\log(d)/\varepsilon)$
Cormode et al. [CPST12]	$O(k \log(d+u))$	$O(1)$	$O(\log(d)/\varepsilon)$	$O(\log(d)/\varepsilon)$
Balcer & Vadhan [BV19]	$\tilde{O}\left(\frac{n}{\varepsilon} \log(d)\right)$	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$	$O\left(\frac{1}{\varepsilon}\right)$	$O(\log(d)/\varepsilon)$
Theorem 5.10 (this work)	$O(k \log(d+u))$	$O(\log(d))$	$O\left(\frac{1}{\varepsilon}\right)$	$O(\log(d)/\varepsilon)$
Korolova et al. [KKMN09]	$O(k \log(d+u))$	$O(1)$	$O(\log(1/\delta)/\varepsilon)$	$O(\log(1/\delta)/\varepsilon)$
Theorem 5.11 (this work)	$O(k(\log(d+u) + \log(1/\delta)))$	$O(\log(1/\delta))$	$O\left(\frac{1}{\varepsilon}\right)$	$O(\log(1/\delta)/\varepsilon)$

TABLE 2. Comparison with previous work. The performance is stated for worst-case input, and all bounds hold in expectation. Space for storing hash functions is not considered. The first four rows are results on ε-differential privacy, and the last two are on (ε, δ)-differential privacy. The $\tilde{O}$-notation suppresses logarithmic factors.

with probability p and answer truthfully with probability $1 - p$. We define randomized response for a Boolean value $b \in \{0, 1\}$ as follows:

$$\text{RandResponse}\,(b, p) = \begin{cases} 1 - b & \text{with probability } p \\ b & \text{with probability } 1 - p \end{cases}$$

Universal Hashing. A *hash family* is a collection of functions $\mathcal{H}$ mapping keys from a universe U to a range R. A family $\mathcal{H}$ is called *universal* if each pair of different keys collides with probability at most $1/|R|$, where the randomness is taken over the random choice of $h \in \mathcal{H}$. A particularly efficient construction that uses $O(\log |U|)$ bits and constant evaluation time is presented in [DHKP97].

Model of Computation. We use the w-bit word RAM model defined by Hagerup [Hag98] where $w = \Theta(\log(d) + \log(u))$. This model allows constant time memory access and basic operations on w-bit words. As such, we can store a k-sparse vector using $O(k \log(d+u))$ bits with constant lookup time using a hash table. We assume that the privacy parameters ε and δ can be represented in a single word.

Negative Values. In this paper, we consider vectors with nonnegative real values, but the mechanism can be generalized for negative values using the following reduction. Let $v \in \mathbb{R}^d$ be a real-valued k-sparse vector. Construct $x, y \in \mathbb{R}^d_+$ from v such that $x_i = \max(v_i, 0)$ and $y_i = -\min(v_i, 0)$. By construction both x and y are k-sparse and the ℓ_1-distance between vectors is preserved. We can access elements in v as $v_i = x_i - y_i$. As such, any differentially private representation of x and y can be used as a differentially private representation of v with at most twice the error. We can avoid the increased error by instead increasing the access time. We discuss this variant of our mechanism in Section 10.

3. Related work

Previous work on releasing differentially private sparse vectors primarily focused on the special case of discrete vectors in the context of releasing the histogram of a dataset.

Korolova, Kenthapadi, Mishra, and Ntoulas [KKMN09] first introduced an approximately differentially private mechanism for the release of a sparse histogram. A similar mechanism was later introduced independently by Bun, Nissim, and Stemmer [BNS19] in another context. The mechanism adds noise to non-zero entries and removes those with a noisy value below a threshold $t = O(\log(1/\delta)/\varepsilon)$. The threshold is chosen such that the probability of releasing an entry with true value 1 is at most δ. The expected maximum error is $O(\log(\max(k, 1/\delta))/\varepsilon)$. Since δ is usually chosen to be negligible in the input size, we assume that $\delta \leq 1/k$. As such, the expected maximum error is $O(\log(1/\delta)/\varepsilon)$. We discuss the per-entry error below. Their mechanism is designed to satisfy differential privacy for discrete data. We extend their technique to real-valued data as part of Section 5, where we combine it with our mechanism.

Cormode, Procopiuc, Srivastava, and Tran [CPST12] introduced a differentially private mechanism in their work on range queries for sparse data. The mechanism adds noise to all entries and removes those with a noisy value below a threshold $t = O(\log d/\varepsilon)$. Here the threshold is used to reduce the expected output size. The number of noisy entries above t is $O(k)$ with high probability. The construction time of a naive implementation of their technique scales linearly in d. They improve on this by sampling from a binomial distribution to determine the number of zero entries to store. They show that their approach produces the same output distribution as a naïve implementation that adds noise to every entry. Their mechanism works for real-valued data in a straightforward way.

Since the expected number of non-zero entries in the output is $O(k)$ for both mechanisms above, their memory requirement is $O(k \log(d + u))$ bits using a hash table. An entry is accessed in constant time. The expected per-entry error depends on the true value of the entry. If the noisy value is above the threshold with sufficiently high probability, the expected error is $O(1/\varepsilon)$. However, this does not hold for entries that are likely removed. Consider for example an entry with a true value exactly at the threshold t. This entry is removed for any negative noise added. As such the expected per-entry error is $O(t)$ for worst-case input, which is $O(\log(1/\delta)/\varepsilon)$ and $O(\log(d)/\varepsilon)$ for the two mechanisms, respectively.

In their work on differential privacy on finite computers, Balcer and Vadhan [BV19] introduced several algorithms including some with similar utility as the mechanisms described above. Moreover, they provided a lower bound of $\Omega(\min\{\log(d), \log(\varepsilon/\delta), n\}/\varepsilon)$ for the expected per-entry error of any algorithm that always outputs a sparse histogram. (See [BV19, Theorem 7.2] for the precise technical statement.) Here n is the number of rows in the dataset, i.e., the sum of all entries of the histogram. This lower bound means that an algorithm that always outputs a $O(k)$-sparse histogram cannot achieve $O(1/\varepsilon)$ expected per-entry error for all input. They bypass this bound by producing a compact representation of a dense histogram. Their representation has expected per-entry and maximum error of $O(1/\varepsilon)$ and $O(\log(d)/\varepsilon)$, respectively. It requires $\tilde{O}(n \log(d)/\varepsilon)$ bits and an entry is accessed in time $\tilde{O}(n/\varepsilon)$. Note that their problem setup differs from ours in that each entry is bounded only by n such that $\|x\|_\infty \leq n$. That is, n serves a similar purpose as u does in our setup. We do not know how to extend their approach to our setup with real-valued input.

In light of the results achieved in previous work, our motivation is to design a mechanism that achieves three properties simultaneously: $O(1/\varepsilon)$ expected per-entry error for arbitrary

Algorithm 2: ALP1-Projection

Parameters : $\alpha, \beta > 0$, and $s \in \mathbb{N}$.
Input : k-sparse vector $x \in \mathbb{R}_+^d$ where $s > 2k$. Sequence of universal hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_m)$, where $m = \lceil \beta/\alpha \rceil$.
Output : 1-differentially private representation of x.

(1) Apply random rounding to a scaled version of each non-zero entry of x such that $y_i = \text{RandRound}\,(x_i/\alpha)$.
(2) Construct $z \in \{0,1\}^{s \times m}$ by hashing the unary representations of y such that:

$$z_{a,b} = \begin{cases} 1, & \exists i : \ b \leq y_i \text{ and } h_b(i) = a \\ 0, & \text{otherwise} \end{cases}$$

(3) Apply randomized response to each bit of z such that $\tilde{z}_{a,b} = \text{RandResponse}\,(z_{a,b}, 1/(\alpha+2))$.
(4) Release h and $\tilde{z}$.

input, fast access, and (asymptotically) optimal space. Previous approaches achieved only at most two of these properties simultaneously. Moreover, we want the per-entry error to match the tail bounds of the Laplace mechanism up to constant factors. We construct a compact representation of a dense vector to bypass the lower bound for sparse vectors by Balcer and Vadhan [BV19]. The access time of our mechanism is $O(\log(d))$ and $O(\log(1/\delta))$ for pure and approximate differential privacy, respectively. Table 2 summarizes the results of previous work and our approach.

4. The ALP mechanism

In this section, we introduce the Approximate Laplace Projection (ALP) mechanism[1] and give an upper bound on the expected per-entry error. The ALP mechanism consists of two algorithms. The first algorithm constructs a differentially private representation of a k-sparse vector and the second estimates the value of an entry based on its representation.

4.1. A 1-differentially private algorithm.

We start by considering the special case of $\varepsilon = 1$ and later generalize to all values of $\varepsilon > 0$. Moreover, the mechanism works well only for entries bounded by a parameter β. In general, this would mean that we had to set $\beta = u$ if we only were to use the ALP mechanism. However, in Section 5 we discuss how to set β smaller and still perform well for all entries.

In the first step of the projection algorithm, we scale every non-zero entry by a parameter of the algorithm and use random rounding to map each such entry to an integer. We then store the unary representation of these integers in a two-dimensional bit-array using a sequence of universal hash functions [CW79]. We call this bit-array the *embedding*. Lastly, we apply randomized response on the embedding to achieve privacy. The pseudocode of the algorithm is given in Algorithm 2, and we discuss it next.

[1]The name is chosen to indicate that the error distribution is approximately like the Laplace distribution, and that we *project* the sparse vector to a much lower-dimensional representation. It also celebrates the mountains, whose silhouette plays a role in a certain random walk considered in the analysis of the ALP mechanism.

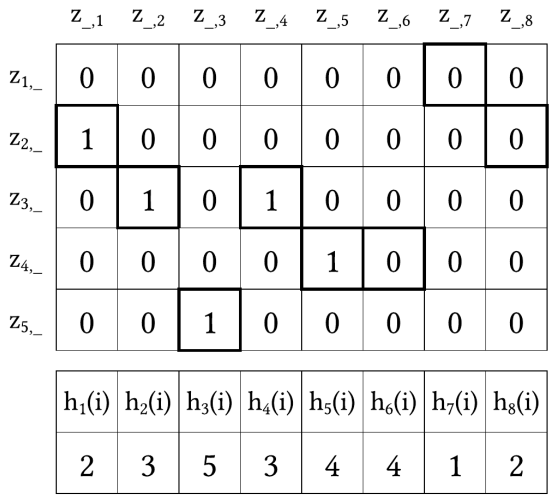


FIGURE 1. Embedding with $\beta/\alpha = m = 8$, $s = 5$ and $y_i = 5$. The ith entry is the only non-zero entry.

Figure 1 shows an example of an embedding before applying randomized response. The input is a vector x where the ith entry x_i is the only non-zero value. The result of evaluating i for each hash function is shown in the table at the bottom and the $m = 8$ bits representing the ith entry in the bit-array are highlighted. In Step (1) of the algorithm, x_i is scaled by $1/\alpha$ and randomized rounding is applied to the scaled value. This results in $y_i = 5$. Using the hash functions, we represent this value in unary encoding by setting the first five bits to 1 in Step (2), where the jth bit is selected by evaluating the hash function h_j on i. The final three bits are unaffected by the entry. Finally, we apply randomized response in each cell of the bit-array. The bit-array after applying randomized response is not shown here, but we present it later in Figure 2. Both the bit-array and the hash functions are the differentially private representation of the input vector x. We use this construction later when estimating the value of x_i.

The algorithm takes three parameters α, β, and s. The parameters α and s are adjustable and affect constant factors for space usage, error, and access time. By increasing s, we reduce the probability of hash collisions while increasing the size of the representation. The parameter α is used to balance two sources of error: by lowering α, we reduce the error in the encoding in Step (1), but increase the noise in Step (3). This parameter also affects the size because it is used to set m. We further discuss these parameters later as part of the error analysis. In Section 9 we discuss how to select values for α and s. Throughout the paper we sometimes assume that α is a constant and s is a constant multiple of k, that is, $\alpha = \Theta(1)$ and $s = \Theta(k)$. The parameter β bounds the values stored in the embedding. We discuss β as part of the error analysis as well.

Lemma 4.1. *Algorithm 2 satisfies 1-differential privacy.*

Proof. We prove the lemma in several steps. Let $x, x' \in \mathbb{R}_+^d$ denote two neighboring vectors.

First, the vectors differ only in their ith entry. In this case, we start by assuming that only a single bit of z is affected by changing x to x' and that there are no hash collisions.

We then allow z to differ in several bits and include hash collisions. Finally, we generalize to the case where x and x' differ in more than one entry.

Assume that z differs only in a single bit for x and x'. Let Y and Y' denote the events that the affected bit is set to one after running the algorithm with input x and x', respectively. Let $p = 1/(\alpha + 2)$ be the parameter of the randomized response step. Then we have $\Pr[Y] = (1 - r) \cdot p + r \cdot (1 - p)$, where $r = x_i/\alpha - \lfloor \min(x_i, x'_i)/\alpha \rfloor$ denotes the probability of the bit being one before the randomized response step. Similarly for x' we define $r' = x'_i/\alpha - \lfloor \min(x_i, x'_i)/\alpha \rfloor$. The minimum term is needed when $\max(x_i, x'_i)$ is a multiple of α such that $\max(r, r') = 1$. We find the difference in the probability of Y and Y' occurring as

$$\begin{aligned}\Pr[Y] - \Pr[Y'] &= (p + r - 2rp) - (p + r' - 2r'p) \\ &= (r - r') \cdot (1 - 2p) \\ &= \frac{x_i - x'_i}{\alpha + 2} \ .\end{aligned}$$

By symmetry, the absolute difference in probability for setting the bit to either zero or one is $|x_i - x'_i|/(\alpha + 2)$. Let Z be an arbitrary output of Algorithm 2. Since x and x' agree on all but the ith entry, the change in probability of outputting Z depends only on the affected bit. Now let Y and Y' denote the events that the bit agrees with output Z for input x and x'. Then we find the ratio of probabilities of outputting Z as

$$\begin{aligned}\frac{\Pr[\text{ALP1-Projection}(x') = Z]}{\Pr[\text{ALP1-Projection}(x) = Z]} = \frac{\Pr[Y']}{\Pr[Y]} &\leq \frac{\Pr[Y] + \frac{|x_i - x'_i|}{\alpha+2}}{\Pr[Y]} \\ &\leq \frac{p + \frac{|x_i - x'_i|}{\alpha+2}}{p} = 1 + |x_i - x'_i| \\ &\leq e^{|x_i - x'_i|} \ .\end{aligned}$$

Here the second inequality follows from $p \leq \Pr[Y] \leq 1 - p$. Next, we take hash collisions into account as follows: Let p' denote the probability that the bit agrees with Z for input x after setting the ith entry to zero. That is, we have $p \leq p' \leq 1 - p$ and $\Pr[Y] = (1 - r) \cdot p' + r \cdot (1 - p)$. The absolute difference in probability remains bounded such that $\Pr[Y] - \Pr[Y'] \leq |x_i - x'_i|/(\alpha + 2)$. Therefore, it still holds that

$$\frac{\Pr[\text{ALP1-Projection}(x') = Z]}{\Pr[\text{ALP1-Projection}(x) = Z]} \leq e^{|x_i - x'_i|} \ .$$

Finally, we remove the assumption that only a single bit of z is affected by composing probabilities. We provide the following inductive construction. Let $x, x' \in \mathbb{R}^d_+$ be vectors that differ in the ith entry such that exactly two bits of z are affected. We consider the case of $x_i < x'_i$ and fix a vector $x'' \in \mathbb{R}^d_+$ with $x_i < x''_i < x'_i$ such that the differences affects exactly one bit each. Again, let Z be an arbitrary output of Algorithm 2. Applying the upper bound from above twice, we may bound the change in probabilities by:

$$\begin{aligned}\frac{\Pr[\text{ALP1-Projection}(x') = Z]}{\Pr[\text{ALP1-Projection}(x) = Z]} &= \frac{\Pr[\text{ALP1-Projection}(x'') = Z]}{\Pr[\text{ALP1-Projection}(x) = Z]} \\ &\quad \cdot \frac{\Pr[\text{ALP1-Projection}(x') = Z]}{\Pr[\text{ALP1-Projection}(x'') = Z]} \\ &\leq e^{|x_i - x_i''|} \cdot e^{|x_i'' - x_i'|} \\ &= e^{|x_i - x_i'|} \ ,\end{aligned}$$

which can be applied inductively if changing an entry affects more than two bits of z.

We are now ready to generalize to any vectors $x, x' \in \mathbb{R}_+^d$, i.e., where vectors may differ in more than a single position. Using the bound from above, we can bound the ratio of probabilities by:

$$\begin{aligned}\frac{\Pr[\text{ALP1-Projection}(x') = Z]}{\Pr[\text{ALP1-Projection}(x) = Z]} &\leq \prod_{i \in [d]} e^{|x_i - x_i'|} \\ &= e^{\sum_{i \in [d]} |x_i - x_i'|} \\ &= e^{\|x - x'\|_1} \ .\end{aligned}$$

The privacy loss is thus bounded by the ℓ_1-distance of the vectors for any output. Recall that the ℓ_1-distance is bounded above by 1 for two neighboring vectors. As such the algorithm is 1-differentially private as for any pair of neighboring vectors x and x' and any subset of outputs S we have:

$$\begin{aligned}\Pr[\text{ALP1-Projection}(x) \in S] &\leq e^{\|x - x'\|_1} \Pr[\text{ALP1-Projection}(x') \in S] \\ &\leq e \cdot \Pr[\text{ALP1-Projection}(x') \in S] \ .\end{aligned}$$

□

The following lemma summarizes the space complexity of storing the bit-array and the collection of hash functions.

Lemma 4.2. *The number of bits required to store h and $\tilde{z}$ is*

$$O\left(\frac{(s + \log d) \cdot \beta}{\alpha}\right) \ .$$

Proof. By definition $m = O(\beta/\alpha)$, so that $s \cdot m = O(s\beta/\alpha)$ bits are used to store $\tilde{z}$. Each hash function uses $O(\log(d))$ bits for a total of $O(\log(d)\beta/\alpha)$ bits to store h. □

4.2. **Estimating an entry.** We now introduce the algorithm to estimate an entry based on the embedding from Algorithm 2. When accessing the ith entry, we estimate the value of y_i and multiply by α to reverse the initial scaling of x_i. The estimate of y_i is chosen to maximize a partial sum. If multiple values maximize the sum, we use their average.

Algorithm 3: ALP1-Estimator

Parameters: $\alpha > 0$.
Input : Embedding $\tilde{z} \in \{0,1\}^{s \times m}$. Sequence of universal hash functions $h = (h_1, \ldots, h_m)$. Index $i \in [d]$.
Output : Estimate of x_i.

(1) Define the function $f\colon \{0, \ldots, m\} \to \mathbb{Z}$ as:

$$f(n) = \sum_{a=1}^{n} 2\tilde{z}_{h_a(i),a} - 1$$

(2) Let P be the set of arguments maximizing f. That is,

$$P = \{n \in \{0, \ldots, m\} : f(a) \leq f(n) \text{ for all } a \in \{0, \ldots, m\}\}$$

(3) Let $\tilde{y}_i = \text{average}(P)$
(4) Return $\tilde{y}_i \cdot \alpha$.

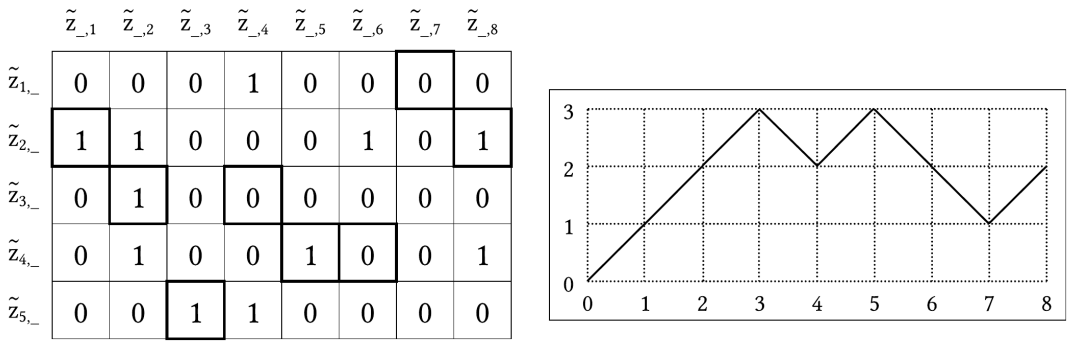


FIGURE 2. Estimation of ith entry from Figure 1.
The partial sum is maximized at indices 3 and 5.
The estimate is 4, while the true value was 5.

Intuition. The first y_i bits representing the ith entry are set to one before applying noise in Algorithm 2, cf. Figure 1. The last $m - y_i$ bits are zero, except if there are hash collisions. Some bits might be flipped due to randomized response, but we expect the majority of the first y_i bits to be ones and the majority of the remaining $m - y_i$ bits to be zeros. The estimate of y_i is based on prefixes maximizing the difference between ones and zeros. The pseudocode for the algorithm is given as Algorithm 3.

Figure 2 shows an example of Algorithm 3. The example is based on the embedding from Figure 1 after adding noise. The plot shows the value of f for all candidate estimates. This sum is maximized at positions 3 and 5, and is visualized as the global *peaks* in the plot. The estimate is the average of those positions.

Lemma 4.3. *The evaluation time of Algorithm 3 is $O(\beta/\alpha)$.*

Proof. We can compute all partial sums by evaluating each bit $(\tilde{z}_{h_1(i),1}, \ldots, \tilde{z}_{h_m(i),m})$ once using dynamic programming. Thus, the evaluation time is $O(m)$ with $m = \lceil \beta/\alpha \rceil$. □

We now analyze the per-entry error of Algorithm 3. We first analyze the expected error based on the parameters of the algorithm. The results are presented in Lemma 4.8. In Lemmas 4.9 and 4.10, we bound the tail distribution of the per-entry error of the algorithm.

Lemma 4.4. *The expected per-entry error of Algorithm 3 is bounded by* $(1/2+\mathbb{E}[|y_i-\tilde{y}_i|])\cdot\alpha$ *for entries with a value of at most* β*.*

Proof. It is clear that the error of the ith entry is α times the difference between $\tilde{y}_i$ and $\frac{x_i}{\alpha}$. The expected difference is bounded by:

$$\mathbb{E}\left[\left|\frac{x_i}{\alpha}-\tilde{y}_i\right|\right] \leq \mathbb{E}\left[\left|\frac{x_i}{\alpha}-y_i\right|\right] + \mathbb{E}[|y_i-\tilde{y}_i|] \leq \frac{1}{2}+\mathbb{E}[|y_i-\tilde{y}_i|] \ .$$

The last inequality follows from Lemma 2.6. □

We find an upper bound on $\mathbb{E}[|y_i-\tilde{y}_i|]$ by analyzing simple random walks. A simple random walk is a stochastic process such that $S_0=0$ and $S_n=\sum_{\ell=1}^{n}X_\ell$, where X are independent and identically distributed random variables with $\Pr[X_\ell=1]=p$ and $\Pr[X_\ell=-1]=1-p$.

Lemma 4.5. *Let S be a simple random walk with $p<1/2$. At any step n the probability that there exists a later step $\ell>n$ such that $S_\ell>S_n$ is $p/(1-p)$.*

Proof. Alm [Alm02, Theorem 1] showed that for any $p<1/2$ there exists a step $\ell>n$ such that $S_\ell=S_n+k$ with probability $[(p/(1-p)]^k$. The Lemma follows from the case of $k=1$. □

For our analysis, we are concerned with the maximum n such that $S_n\geq 0$. For an infinite random walk where $p<1/2$ such an n exists with probability 1.

Lemma 4.6. *Let S be a simple random walk with $p<1/2$. The expected last nonnegative step of S is:* $\mathbb{E}[\max_n : S_n\geq 0]=4(p-p^2)/(1-2p)^2$.

Proof. We use Lemma 4.5 to find the probability that S_n is the unique maximum in $\{S_n,\ldots,S_\infty\}$ as follows:

$$\begin{aligned}\Pr[S_n>\max(\{S_{n+1},\ldots,S_\infty\})] &= \Pr[X_{n+1}=-1]\cdot\Pr[S_{n+1}=\max(\{S_{n+1},\ldots,S_\infty\})]\\ &= (1-p)\cdot\left(1-\frac{p}{1-p}\right)=1-2p \ .\end{aligned}$$

The last nonnegative step must have value exactly zero and as such must be at an even numbered step. The probability that step $2i$ is the last nonnegative is:

$$\begin{aligned}\Pr[(\max_n : S_n\geq 0)=2i] &= \Pr[S_{2i}=0]\cdot\Pr[S_i>\max(\{S_{i+1},\ldots,S_\infty\})]\\ &= \binom{2i}{i}p^i(1-p)^i(1-2p)=\binom{2i}{i}(p-p^2)^i(1-2p) \ .\end{aligned}$$

We are now ready to find the expected last nonnegative step of an infinite simple random walk as

$$\begin{aligned}\mathbb{E}[\max_n : S_n\geq 0] &= \sum_{i=0}^{\infty}2i\cdot\Pr[(\max_n : S_n\geq 0)=2i]=\sum_{i=0}^{\infty}2i\binom{2i}{i}(p-p^2)^i(1-2p)\\ &= 2(1-2p)\sum_{i=0}^{\infty}i\binom{2i}{i}(p-p^2)^i=\frac{4(p-p^2)}{(1-2p)^2} \ .\end{aligned}$$

The last equality follows from the identity $\sum_{i=0}^{\infty} i\binom{2i}{i}(p-p^2)^i = 2(p-p^2)/(1-2p)^3$. See Appendix A for a proof of this identity. □

We are now ready to bound $\mathbb{E}[|y_i - \tilde{y}_i|]$. We consider entries with value at most β, i.e., $y_i \leq m$.

Lemma 4.7. *Let $y_i \leq m$ and $\gamma = (\alpha+2)/(1+\alpha k/s) - 2$. Then the expected value of $|y_i - \tilde{y}_i|$ is bounded such that*

$$\mathbb{E}[|y_i - \tilde{y}_i|] \leq \frac{4\alpha+4}{\alpha^2} + \frac{4\gamma+4}{\gamma^2} \ .$$

Proof. Recall the definition of P from Algorithm 3. Let $\bar{y}_i \in P$ denote an element furthest from y_i, that is, $|y_i - a| \leq |y_i - \bar{y}_i|$ for all $a \in P$. It it clearly sufficient to consider $\bar{y}_i$ for the proof since $|y_i - \tilde{y}_i| \leq |y_i - \bar{y}_i|$. We first consider the case of $\bar{y}_i \leq y_i$. It follows from the definition of $\bar{y}_i$ as a maximum that $\sum_{j=\bar{y}_i+1}^{y_i} \tilde{z}_{h_j(i),j} \leq 0$. As such at least half the bits $(\tilde{z}_{h_{\bar{y}_i+1}(i),\bar{y}_i+1}, \ldots, \tilde{z}_{h_{y_i}(i),y_i})$ must be zero, that is they were flipped by randomized response in Step (3) of Algorithm 2. As such the length of the longest interval ending at bit $\tilde{z}_{h_{y_i}(i),y_i}$ where at least half the bits were flipped is an upper bound on the value of $y_i - \bar{y}_i$. The expected size of said interval is bounded by the expected last nonnegative step of a simple random walk with $p = 1/(\alpha+2)$. It follows from Lemma 4.6 that

$$\mathbb{E}[y_i - \bar{y}_i \mid \ \bar{y}_i \leq y_i] \leq \frac{4(p-p^2)}{(1-2p)^2} = \frac{\frac{4\alpha+4}{(\alpha+2)^2}}{\frac{\alpha^2}{(\alpha+2)^2}} = \frac{4\alpha+4}{\alpha^2} \ .$$

We can use a similar argument when $y_i \geq \bar{y}_i$ to show that at least half the bits in $(\tilde{z}_{h_{y_i+1}(i),y_i+1}, \ldots, \tilde{z}_{h_{\bar{y}_i}(i),\bar{y}_i})$ must be 1 since $\bar{y}_i$ is a maximum. In this case we have to consider the possibility of hash collisions. Each hash function maps to $[s]$ and at most k entries result in a hash collision. The probability of a hash collision is at most k/s using a union bound. Consequently, for $j > y_i$, $\Pr[\tilde{z}_{h_j(i),j} = 1] \leq (1-k/s)\cdot p + k/s \cdot (1-p) = (1+\alpha k/s)/9\alpha + 20$. We let $(1+\alpha k/s)/(\alpha+2) = 1/(\gamma+2)$ such that $\mathbb{E}[\bar{y}_i - y_i \mid \bar{y}_i \geq y_i] \leq (4\gamma+4)/\gamma^2$ by Lemma 4.6 and the calculation above. We isolate γ to find

$$\frac{1}{\gamma+2} = \frac{1+\frac{\alpha k}{s}}{\alpha+2}$$
$$(\Leftrightarrow) \qquad \gamma = \frac{\alpha+2}{1+\frac{\alpha k}{s}} - 2 \ .$$

Note that $\gamma > 0$ holds due to the requirement $s > 2k$ of Algorithm 2. By conditional expectation, we obtain an upper bound on the total expected error:

$$\mathbb{E}[|y_i - \tilde{y}_i|] \leq \mathbb{E}[|y_i - \bar{y}_i|] \leq \mathbb{E}[y_i - \bar{y}_i \mid \bar{y}_i \leq y_i] + \mathbb{E}[\bar{y}_i - y_i \mid \ \bar{y}_i \geq y_i] \ \leq \frac{4\alpha+4}{\alpha^2} + \frac{4\gamma+4}{\gamma^2} \ . \tag{4.1}$$

□

Therefore, we can bound the expected per-entry error for entries with a true value of at most β by a function of the parameters α and s. In Section 9 we discuss the choice of these parameters based on the upper bond and experiments. For any fixed values of α and k/s we have:

Lemma 4.8. *Let $\alpha = \Theta(1)$ and $s = \Theta(k)$. Then the expected per-entry error of Algorithm 3 is $\mathbb{E}[|x_i - \tilde{x}_i|] \leq \max(0, x_i - \beta) + O(1)$.*

Proof. It follows from Lemmas 4.4 and 4.7 that the expected error for any entry bounded by β satisfies:

$$x_i \leq \beta \implies \mathbb{E}[|x_i - \tilde{x}_i|] \leq \left(\frac{1}{2} + \frac{4\alpha + 4}{\alpha^2} + \frac{4\gamma + 4}{\gamma^2}\right) \cdot \alpha \ ,$$

where $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$. Entries exceeding β have an additional error of up to $x_i - \beta$, since $y_i = m$ and $y_i > m$ are represented identically in the embedding by Algorithm 2. Since α and k/s are constants,

$$\mathbb{E}[|x_i - \tilde{x}_i|] \leq \max(0, x_i - \beta) + O(1) \ .$$

□

Next, we bound the tail probabilities for the per-entry error of the mechanism. We bound the error of the estimate $\tilde{y}_i$, which implies bounds on the error of the mechanism.

Lemma 4.9. *Let $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$ and suppose that $\tau > 0$. Let $p = 1/(\gamma + 2)$. For any fixed index $i \in [d]$ when using Algorithm 3 we have*

$$\Pr[|y_i - \tilde{y}_i| \geq \tau] \leq \frac{2 \cdot (4(p - p^2))^{\tau/2}}{\sqrt{\pi}(1 - 2p)} \ ,$$

Proof. Let S be a simple random walk. We find an upper bound on the probability that the position of the last non-negative step in S is at least τ:

$$\begin{aligned}\Pr[(\max_n : S_n \geq 0) \geq \tau] &= \sum_{j=\lceil \tau/2 \rceil}^{\infty} \binom{2j}{j} (p - p^2)^j (1 - 2p) \leq \frac{1 - 2p}{\sqrt{\pi}} \sum_{j=\lceil \tau/2 \rceil}^{\infty} (4(p - p^2))^j \\ &= \frac{1 - 2p}{\sqrt{\pi}} \frac{(4(p - p^2))^{\lceil \tau/2 \rceil}}{1 - 4(p - p^2)} \leq \frac{(4(p - p^2))^{\tau/2}}{\sqrt{\pi}(1 - 2p)} \ ,\end{aligned}$$

where the first inequality follows from $\binom{2j}{j} \leq 4^j/\sqrt{\pi j}$ when $j \geq 1$ [Elk13]. The last inequality simply follows from $1 - 4(p - p^2) = (1 - 2p)^2$ and $4(p - p^2) < 1$. As discussed in the proof of Lemma 4.7, the expectation of $|y_i - \tilde{y}_i|$ can be bounded by two random walks each with p at most $1/(\gamma + 2)$. □

Lemma 4.10. *Let $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$ and $p = 1/(\gamma + 2)$. For any fixed index $i \in [d]$ with probability at least $1 - \psi$ for Algorithm 3,*

$$|y_i - \tilde{y}_i| \leq \frac{2 \log\left(\frac{2}{\psi\sqrt{\pi}(1 - 2p)}\right)}{\log(1/(4p - 4p^2))} \ .$$

Proof. We set $\psi = 2 \cdot (4(p - p^2))^{\tau/2}/\sqrt{\pi}(1 - 2p)$ and isolate τ such that:

$$\tau = \frac{2 \log\left(\frac{2}{\psi\sqrt{\pi}(1 - 2p)}\right)}{\log(1/(4p - 4p^2))} \ .$$

By Lemma 4.9 we have: $\Pr[|y_i - \tilde{y}_i| \leq \tau] \geq 1 - \psi$. □

Algorithm 4: ALP-Projection

Parameters: $\alpha, \beta, \varepsilon > 0$, and $s \in \mathbb{N}$.
Input: k-sparse vector $x \in \mathbb{R}_+^d$, where $s > 2k$. Sequence of universal hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_m)$, where $m = \lceil \beta\varepsilon/\alpha \rceil$.
Output: ε-differentially private representation of x.

(1) Scale the entries of x such that $\hat{x}_i = x_i \cdot \varepsilon$.
(2) Let $h, \tilde{z} = \text{ALP1-Projection}_{\alpha,\beta\cdot\varepsilon,s}(\hat{x}, h)$.
(3) Release h and $\tilde{z}$.

Up to constant factors, the tail probabilities of our mechanism are similar to the properties of the Laplace mechanism summarized in Proposition 2.5. The probabilities depend on the parameters of the mechanism. In Section 9, we fix the parameters and evaluate the error in practice. We summarize the tail probabilities for $|x_i - \tilde{x}_i|$ in Lemma 4.11.

Lemma 4.11. *Let $\gamma = (\alpha+2)/(1+\alpha k/s) - 2$ and $p = 1/(\gamma+2)$, and assume that $x_i \le \beta$ and $\tau \ge \alpha$. For any fixed index $i \in [d]$ when using Algorithm 3,*

$$\Pr[|x_i - \tilde{x}_i| \ge \tau] < \frac{2 \cdot (4(p-p^2))^{(\tau/2\alpha)-1/2}}{\sqrt{\pi}(1-2p)} \ ,$$

Moreover, with probability at least $1-\psi$,

$$|x_i - \tilde{x}_i| < \left(1 + \frac{2\log\left(\frac{2}{\psi\sqrt{\pi}(1-2p)}\right)}{\log(1/(4p-4p^2))}\right) \cdot \alpha \ .$$

Proof. It is easy to see that $|x_i - x_i'| < (1 + |y_i - \tilde{y}_i|) \cdot \alpha$ holds, as the error of random rounding is strictly less than 1. The bounds follow from Lemmas 4.9 and 4.10. □

4.3. **Generalization to ε-differential privacy.** We now generalize the ALP mechanism from 1-differential privacy to satisfying ε-differential privacy. A natural approach is to use a function of ε as the parameter for randomized response in Algorithm 2. The projection algorithm is ε-differentially private if we remove the scaling step and set $p = 1/(\varepsilon+2)$. However, the expected per-entry error would be bounded by $(8\varepsilon+8)/\varepsilon^2$ by Equation 4.1 (without considering hash collisions), which is as large as $O(1/\varepsilon^2)$ for small values of ε. Other approaches modifying the value of p have a similar expectation. In Section 7 we discuss a special case in which such an approach is useful for large ε.

In the following, we use simple pre-processing and post-processing steps to achieve optimal error. The idea is to scale the input vector as well as the parameter β by ε before running Algorithm 2. We scale back the estimates from Algorithm 3 by $1/\varepsilon$. These generalizations are given as Algorithm 4 and Algorithm 5, respectively.

Lemma 4.12. *Algorithm 4 satisfies ε-differential privacy.*

Proof. It follows from the proof of Lemma 4.1 that for any subset of outputs S we have $\Pr[\text{ALP1-Projection}(\hat{x}') \in S] / \Pr[\text{ALP1-Projection}(\hat{x}) \in S] \le e^{\|\hat{x}-\hat{x}'\|_1}$. Therefore, for any

Algorithm 5: ALP-Estimator

Parameters: $\alpha, \varepsilon > 0$.
Input : Embedding $\tilde{z} \in \{0,1\}^{s \times m}$. Sequence of universal hash functions $h = (h_1, \ldots, h_m)$. Index $i \in [d]$.
Output : Estimate of x_i.

(1) Let $\tilde{x}_i = \text{ALP1-Estimator}_\alpha(\tilde{z}, h, i)$.
(2) Return $\frac{\tilde{x}_i}{\varepsilon}$.

pair of neighboring vectors x and x',

$$\frac{\Pr[\text{ALP-Projection}(x') \in S]}{\Pr[\text{ALP-Projection}(x) \in S]} = \frac{\Pr[\text{ALP1-Projection}(\hat{x}') \in S]}{\Pr[\text{ALP1-Projection}(\hat{x}) \in S]} \leq e^{\|\hat{x}-\hat{x}'\|_1} = e^{\varepsilon \cdot \|x-x'\|_1} \leq e^{\varepsilon} \ .$$

□

Lemma 4.13. *Let $\alpha = \Theta(1)$ and $s = \Theta(k)$. The output of Algorithm 4 can be stored using $O((k + \log d)\beta\varepsilon)$ bits.*

Proof. It follows directly from Lemma 4.2. □

Lemma 4.14. *Let $\alpha = \Theta(1)$ and $s = \Theta(k)$. Then the expected per-entry error of Algorithm 5 is $\mathbb{E}[|x_i - \tilde{x}_i|] \leq \max(0, x_i - \beta) + O(1/\varepsilon)$ and the evaluation time is $O(\beta\varepsilon)$.*

Proof. It is clear that the error of Algorithm 5 is $1/\varepsilon$ times the error of Algorithm 3 for entries at most β. As such the expected per-entry error follows from Lemma 4.8. The evaluation time follows directly from Lemma 4.3. □

Lemma 4.15. *Let $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$ and $p = 1/(\gamma + 2)$, and assume that $x_i \leq \beta$ and $\tau \geq \alpha/\varepsilon$. Then any fixed index $i \in [d]$ when using Algorithm 5,*

$$\Pr[|x_i - \tilde{x}_i| \geq \tau] < \frac{2 \cdot (4(p - p^2))^{(\tau\varepsilon/2\alpha)-1/2}}{\sqrt{\pi}(1-2p)} \ ,$$

With probability at least $1 - \psi$ we have:

$$|x_i - \tilde{x}_i| < \left(1 + \frac{2\log\left(\frac{2}{\psi\sqrt{\pi}(1-2p)}\right)}{\log(1/(4p-p^2))}\right) \cdot \frac{\alpha}{\varepsilon} \ .$$

Proof. It follows directly from Lemma 4.11. □

We are now ready to state the following theorem, which summarizes the properties of the ALP mechanism.

Theorem 4.16. *Let $\alpha = \Theta(1)$ and $s = \Theta(k)$. Then there exists an algorithm where the expected per-entry error is $O(1/\varepsilon)$ for all entries, the access time is $O(u\varepsilon)$, and the space usage is $O((k + \log d)u\varepsilon)$ bits.*

Proof. By setting $\beta = u$, it follows directly from Lemmas 4.13 and 4.14. □

Algorithm 6: Threshold [CPST12]

Parameters : $\varepsilon, t > 0$.
Input : k-sparse vector $x \in \mathbb{R}_+^d$.
Output : ε-differentially private representation of x.

(1) Let $v_i = x_i + \eta_i$ for all $i \in [d]$, where $\eta_i \sim \text{Lap}(1/\varepsilon)$.
(2) Truncate entries below t:

$$\tilde{v}_i = \begin{cases} v_i, & \text{if } y_i \geq t \\ 0, & \text{otherwise} \end{cases}$$

(3) Return $\tilde{v}$.

The space usage and access time of the mechanism both scale linearly with the parameter β. As such, the mechanism performs well only for small values of u. However, in many contexts u scales with the input size. One example is a histogram, where u is the number of rows in the underlying dataset. Next, we show how to handle such cases.

5. Combined data structure

In this section, we combine the ALP mechanism with techniques from previous work to improve space requirements and access time. As shown in Theorem 4.16, the ALP mechanism performs well when all entries are bounded by a small value. The per-entry error is low only for entries bounded by β but the space requirements and access time scale linearly with β. Some of the algorithms from previous work perform well for large entries but have large per-entry error for small values. The idea of this section is to combine the ALP mechanism with such an algorithm to construct a composite data structure that performs well for both small and large entries.

To handle large values, we use the thresholding technique from Cormode et al. [CPST12]. It adds noise to each entry, but only stores entries above a threshold. The pseudocode of the algorithm is given as Algorithm 6.

Lemma 5.1. *Algorithm 6 satisfies ε-differential privacy.*

Proof. The algorithm is equivalent to the Laplace mechanism followed by post-processing. The Laplace mechanism satisfies ε-differential privacy, and privacy is preserved under post-processing as stated by Lemma 2.2. □

Lemma 5.2. *Let $t = 2\ln(d)/\varepsilon$. Then the output of Algorithm 6 is k-sparse with probability at least $1 - 1/2d$.*

Proof. Using Definition 2.4, we find that the probability of storing a zero entry of x is:

$$\Pr[\text{Lap}(1/\varepsilon) \geq t] = \Pr[\text{Lap}(1/\varepsilon) \leq -t] = \frac{1}{2}e^{-t\varepsilon} = \frac{1}{2d^2} \ .$$

If the output is not k-sparse there must exists at least one coordinate i such that $\tilde{v}_i \neq 0$ and $x_i = 0$. By a union bound such a coordinate exists with probability at most $1/(2d)$. □

Algorithm 7: Threshold ALP-Projection

Parameters: $\alpha, \varepsilon_1, \varepsilon_2 > 0$, and $s \in \mathbb{N}$.
Input: k-sparse vector $x \in \mathbb{R}_+^d$, where $s > 2k$. Sequence of universal hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_m)$, where $m = \lceil t\varepsilon_2/\alpha \rceil$.
Output: $(\varepsilon_1 + \varepsilon_2)$-differentially private representation of x.

(1) Let $t = 2\ln(d)/\varepsilon_1$.
(2) Let $\tilde{v} = \text{Threshold}_{\varepsilon_1,t}(x)$.
(3) Let $h, \tilde{z} = \text{ALP-Projection}_{\alpha,\varepsilon_2,t,s}(x, h)$
(4) Return $\tilde{v}$, h and $\tilde{z}$.

Algorithm 8: Threshold ALP-Estimator

Parameters: $\alpha, \varepsilon_2 > 0$.
Input: Vector $\tilde{v} \in \mathbb{R}_+^d$. Embedding $\tilde{z} \in \{0,1\}^{s \times m}$. Sequence of universal hash functions $h = (h_1, \ldots, h_m)$. Index $i \in [d]$.
Output: Estimate of x_i.

(1) Estimate the entry using either the vector or the embedding such that:

$$\tilde{x}_i = \begin{cases} \tilde{v}_i, & \text{if } \tilde{v}_i \neq 0 \\ \text{ALP-Estimator}_{\varepsilon_2,\alpha}(\tilde{z}, h, i), & \text{otherwise} \end{cases}$$

(2) Return $\tilde{x}_i$.

As discussed in Section 3, the expected per-entry error of Algorithm 6 is $O(\log(d)/\varepsilon)$ for worst-case input. We combine the algorithm with the ALP mechanism from the previous section to achieve $O(1/\varepsilon)$ expected per-entry error for any input. We use the threshold parameter t as value for parameter β in Algorithm 4. The algorithm is presented in Algorithm 7. We use a separate privacy parameter for each part of the algorithm. Throughout this section we assume that the ratio between them is fixed such that $\varepsilon_1 = \Theta(\varepsilon_2)$. After Lemma 5.5 we discuss what happens if this ratio is not fixed.

Lemma 5.3. *Algorithm 7 satisfies $(\varepsilon_1 + \varepsilon_2)$-differential privacy.*

Proof. The two parts of the algorithm are independent as there is no shared randomness. The first part of the algorithm satisfies ε_1-differential privacy by Lemma 5.1 and the second part satisfies ε_2-differential privacy by Lemma 4.12. As such it follows directly from composition (Lemma 2.3) that Algorithm 7 satisfies $(\varepsilon_1 + \varepsilon_2)$-differential privacy. □

Lemma 5.4. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, $\varepsilon_1 = \Theta(\varepsilon_2)$. Then the output of Algorithm 7 is stored using $O(k\log(d+u) + \log^2(d))$ bits with probability at least $1 - 1/2d$.*

Proof. It follows from Lemma 5.2 that we can store $\tilde{v}$ using $O(k\log(d+u))$ bits with probability at least $1 - 1/2d$. Since $\beta = t$ it follows from Lemma 4.13 that we can store h and $\tilde{z}$ using $O((k + \log(d))t\varepsilon_2) = O(k\log(d) + \log^2(d))$ bits. □

To estimate an entry, we access $\tilde{v}$ when a value is stored for the entry and the ALP embedding otherwise. This algorithm is presented in Algorithm 8.

Lemma 5.5. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon = \varepsilon_1 + \varepsilon_2$ with $\varepsilon_1 = \Theta(\varepsilon_2)$. Let $\tilde{v}$, h, and $\tilde{z}$ be the output of Algorithm 7 given these parameters. Then the evaluation time of Algorithm 8 is $O(\log(d))$. The expected per-entry error is $O(1/\varepsilon)$ and the expected maximum error is $O(\log(d)/\varepsilon)$.*

Proof. The evaluation time follows from Lemma 4.14. That is, the evaluation time is $O(\beta\varepsilon) = O(t\varepsilon) = O(\log(d))$.

The error depends on both parts of the algorithm. The expected per-entry error for the ith entry is $\max(0, x_i - \beta) + O(1/\varepsilon_2)$ when $\tilde{v}_i = 0$ by Lemma 4.14. That is, when η_i is less than $\beta - x_i$ in Algorithm 6. When $\tilde{v}_i \neq 0$ the error is the absolute value of η_i. That is, we can analyze it using conditional probability and the probability density function of the Laplace distribution from Definition 2.4.

$$\begin{aligned}
\mathbb{E}[|x_i - \tilde{x}_i|] &= \mathbb{E}[|x_i - \tilde{x}_i| \mid \tilde{v}_i = 0] \cdot \Pr[\tilde{v}_i = 0] + \mathbb{E}[|x_i - \tilde{x}_i| \mid \tilde{v}_i \neq 0] \cdot \Pr[\tilde{v}_i \neq 0] \\
&\leq (\max(0, x_i - \beta) + O(1/\varepsilon_2)) \cdot \Pr[\mathrm{Lap}(1/\varepsilon_1) < \beta - x_i] \\
&\quad + \int_{\beta - x_i}^{\infty} |v - x_i| \cdot \frac{\varepsilon_1}{2} e^{-|v - x_i|\varepsilon_1} \, dv \\
&< \int_{-\infty}^{\beta - x_i} (|v - x_i| + O(1/\varepsilon_2)) \cdot \frac{\varepsilon_1}{2} e^{-|v - x_i|\varepsilon_1} \, dv \\
&\quad + \int_{\beta - x_i}^{\infty} |v - x_i| \cdot \frac{\varepsilon_1}{2} e^{-|v - x_i|\varepsilon_1} \, dv \\
&< \int_{-\infty}^{\infty} |v - x_i| \cdot \frac{\varepsilon_1}{2} e^{-|v - x_i|\varepsilon_1} \, dv + O(1/\varepsilon_2) \\
&= O(1/\varepsilon_1) + O(1/\varepsilon_2) = O(1/\varepsilon) \ .
\end{aligned}$$

The expected maximum error of Algorithm 6 is $O(\log(d)/\varepsilon)$ and the output of the Algorithm 5 is at most β. Since $\beta = O(\log(d)/\varepsilon)$ the expected maximum error is $O(\log(d)/\varepsilon)$. □

The asymptotic properties of the algorithms hold for any fixed ratio between ε_1 and ε_2. A natural choice is to set $\varepsilon_1 = \varepsilon_2$. However, the ratio affects constant factors, and it might not be the best choice in practice. The value of the parameter m and in turn the space consumption and access time of the projection scales with $\varepsilon_2/\varepsilon_1$. But the parameters also affect the expected error for each part of the algorithm. Furthermore, the constant factor for the error of the Laplace mechanism is lower than that of the projection. Therefore, one could set ε_2 higher than ε_1 to balance out those constant factors. We explore the constant factors of Algorithm 3 further in Section 9.

5.1. **Removing the dependency on dimension.** To make access time independent of the dimension d, we can turn to approximate differential privacy. This allows us to use a smaller threshold in the initial thresholding approach, which in turn results in smaller values for β in the ALP mechanism.

The following algorithm is similar to that introduced by Korolova et al. [KKMN09], which we discussed in Section 3. It adds noise to non-zero entries only, and uses a threshold to satisfy approximate differential privacy. Our algorithm differs from the work of Korolova et al. by using a random rounding step. This step is not needed in a discrete setting, where at most a single zero-valued entry is changed to a non-zero entry for neighboring vectors. However, in the real-valued context, several zero entries can change.

Algorithm 9: Threshold2 (Following technique by [KKMN09])

Parameters : $\varepsilon, \delta > 0$.
Input : k-sparse vector $x \in \mathbb{R}_+^d$.
Output : (ε, δ)-differentially private approximation of x.

(1) Apply random rounding to non-zero entries below 1 such that:

$$y_i = \begin{cases} \text{RandRound}\left(x_i\right), & \text{if } 0 < x_i < 1 \\ x_i, & \text{otherwise} \end{cases}$$

(2) Let $v_i = y_i + \eta_i$ for all non-zero entries, where $\eta_i \sim \text{Lap}(1/\varepsilon)$.
(3) Let $t = \log(1/\delta)/\varepsilon + 2$.
(4) Truncate entries below t:

$$\tilde{v}_i = \begin{cases} v_i, & \text{if } y_i \neq 0 \text{ and } v_i \geq t \\ 0, & \text{otherwise} \end{cases}$$

(5) Return $\tilde{v}$.

Lemma 5.6. *Algorithm 9 satisfies (ε, δ)-differential privacy.*

Proof. Let x and x' be neighboring vectors. We consider two additional vectors $\hat{x}$ and $\hat{x}'$ such that:

$$\hat{x}_i = \begin{cases} \min(1, x'_i), & \text{if } x_i \leq 1 \\ x_i, & \text{otherwise;} \end{cases}$$

$$\hat{x}'_i = \begin{cases} 1, & \text{if } x'_i < 1 \text{ and } 1 < x_i \\ x'_i, & \text{otherwise.} \end{cases}$$

The vectors are constructed such that x and $\hat{x}$ can only differ for entries at most 1 in both vectors. The same holds for x' and $\hat{x}'$. Additionally, the ℓ_1-distance is still at most 1 between any pair of vectors.

We find the probability of outputting anything for an entry less than or equal to 1 to be

$$\begin{aligned} x_i \leq 1 \implies \Pr[\tilde{v}_i \neq 0] &= \Pr[y_i = 1] \cdot \Pr[\text{Lap}(1/\varepsilon) \geq t - 1] = x_i \cdot \Pr[\text{Lap}(1/\varepsilon) \leq -(t-1)] \\ &= x_i \cdot \frac{1}{2} e^{-(t-1)\varepsilon} = x_i \cdot \frac{1}{2} e^{-\ln(1/\delta) - \varepsilon} = \frac{x_i \delta}{2 \cdot e^{\varepsilon}} \ . \end{aligned}$$

Since x and $\hat{x}$ only differ for entries less than or equal to 1, any subset of outputs S,

$$\begin{aligned} \Pr[\text{Threshold2}(x) \in S] &\leq \Pr[\text{Threshold2}(\hat{x}) \in S] + \sum_{i \in [d]} |\hat{x}_i - x_i| \frac{\delta}{2 \cdot e^{\varepsilon}} \\ &\leq \Pr[\text{Threshold2}(\hat{x}) \in S] + \frac{\delta}{2 \cdot e^{\varepsilon}} \ . \end{aligned}$$

The inequality holds in both directions and for the pair of x' and $\hat{x}'$ as well.

By definition $\hat{x}$ and $\hat{x}'$ only differ for entries of at least 1. Consequently, we can ignore the random rounding step, so that

$$\Pr[\text{Threshold2}(\hat{x}) \in S] \leq e^{\|\hat{x} - \hat{x}'\|_1 \varepsilon} \Pr[\text{Threshold2}(\hat{x}') \in S] \leq e^{\varepsilon} \cdot \Pr[\text{Threshold2}(x) \in S] \ .$$

Using the inequalities above we have:

$$\begin{aligned}\Pr[\text{Threshold2}(x) \in S] &\leq \Pr[\text{Threshold2}(\hat{x}) \in S] + \frac{\delta}{2e^{\varepsilon}} \\ &\leq e^{\varepsilon} \cdot \Pr[\text{Threshold2}(\hat{x}') \in S] + \frac{\delta}{2e^{\varepsilon}} \\ &\leq e^{\varepsilon} \cdot \left(\Pr[\text{Threshold2}(x') \in S] + \frac{\delta}{2e^{\varepsilon}}\right) + \frac{\delta}{2e^{\varepsilon}} \\ &\leq e^{\varepsilon} \cdot \Pr[\text{Threshold2}(x') \in S] + \delta \ .\end{aligned}$$

□

Lemma 5.7. *Suppose that $\delta = O(1/k)$. Then the expected maximum error of Algorithm 9 is $O(\log(1/\delta)/\varepsilon)$.*

Proof. The expected maximum error added by the Laplace noise is $O(\log(k)/\varepsilon)$, since we add noise to at most k entries. By removing entries we add error of up to $O(\log(1/\delta)/\varepsilon)$. As such the expected maximum error for worst-case input is:

$$\mathbb{E}[\|x - \tilde{v}\|_{\infty}] \leq O\left(\frac{\log(k)}{\varepsilon}\right) + O\left(\frac{\log(1/\delta)}{\varepsilon}\right) = O\left(\frac{\log(1/\delta)}{\varepsilon}\right) \ .$$

□

In the following, we use Algorithm 9 instead of Algorithm 6 in Step (2) of Algorithm 7.

Lemma 5.8. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, $\varepsilon = \varepsilon_1 + \varepsilon_2$ with $\varepsilon_1 = \Theta(\varepsilon_2)$, and $\delta > 0$ with $\delta = O(1/k)$. By using Algorithm 9 in Algorithm 7 the access time is $O(\log(1/\delta))$. The expected per-entry error is $O(1/\varepsilon)$ and the expected maximum error is $O(\log(1/\delta)/\varepsilon)$. The combined mechanism satisfies (ε, δ)-differential privacy.*

Proof. The proof is the same as the proofs of Lemmas 5.3 and 5.5. □

Lemma 5.9. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon_1 = \Theta(\varepsilon_2)$. Then the memory requirement of combining Algorithm 9 and the ALP mechanism is $O(k \log(d+u) + k \log(1/\delta) + \log(d) \log(1/\delta))$.*

Proof. The output of Algorithm 9 is always k-sparse, and we represent it using $O(k \log(d+u))$ bits. We set $\beta = \ln(1/\delta)/\varepsilon_2 + 2$, and therefore h and $\tilde{z}$ are represented using $O(k \log(1/\delta) + \log(d) \log(1/\delta))$ bits by Lemma 4.13. □

We are now ready to summarize our results for both pure and approximate differential privacy.

Theorem 5.10. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon > 0$. Then there exists an ε-differentially private algorithm with $O(1/\varepsilon)$ expected per-entry error, $O(\log(d)/\varepsilon)$ expected maximum error, access time of $O(\log(d))$, and space usage of $O(k \log(d+u) + \log^2(d))$ with probability at least $1 - 1/2d$.*

Proof. It follows directly from Lemmas 5.3, 5.4 and 5.5. □

Theorem 5.11. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon, \delta > 0$. Then there exists an (ε, δ)-differentially private algorithm with $O(1/\varepsilon)$ expected per-entry error, $O(\log(1/\delta)/\varepsilon)$ expected maximum error, access time of $O(\log(1/\delta))$, and space usage of $O(k \log(d+u) + k \log(1/\delta) + \log(d) \log(1/\delta))$.*

Proof. It follows directly from Lemmas 5.6, 5.8 and 5.9. □

6. Faster evaluation with multiplicative error

From the previous sections, we know how to achieve evaluation time $O(\log d)$ and $O(\log(1/\delta))$, respectively. In this section, we improve the evaluation time to $O\left(\log(\log(d)/\varepsilon)\right)$ and $O\left(\log(\log(1/\delta)/\varepsilon)\right)$ at the cost of a *multiplicative error* $O\left(1\right)$. That is, this technique can be used if it sufficient to estimate the *order of magnitude* of an entry. We first describe the data structure and the estimation algorithm, and then state and analyze its properties.

Projection. Let $\alpha, \varepsilon > 0, B > 1, \beta \geq \max\{1, 1/\ln(B)\}$, and $s \in \mathbb{N}$. Given a k-sparse vector $x \in \mathbb{R}_+^d$ with $x_i \leq \beta$, define $\dot{x} \in \mathbb{R}_+^d$ by $\dot{x}_i = \max\{\log_B(x_i \ln B), 0\}$ for all $x_i > 0$. Run Algorithm 4 with input $\dot{x}$ and a sequence of universal hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_m)$, where $m = \lceil \log_B(\beta \ln B)\varepsilon/\alpha \rceil$. Let $\tilde{z}$ be the output of Algorithm 4.

Estimation. Given $\tilde{z}$, h, and an index $i \in [d]$, let $\ddot{x}_i$ be the output of Algorithm 5 with parameters α and ε. Return the value $\tilde{x}_i = B^{\ddot{x}_i}/\ln B$ as an estimate for x_i.

Lemma 6.1. *The projection algorithm satisfies ε-differential privacy.*

Proof. By Lemma 4.12, Algorithm 4 satisfies ε-differential privacy for neighboring inputs. We have to show that the mapping $x_i \mapsto \max\{\log_B(x_i \ln B), 0\}$ preserves the neighborhood relation for neighboring x and x'. The function $f(x) = \log_B(x \ln B)$ is Lipschitz continuous in $\mathbb{R}_{>1/\ln B}$ since if $x \geq 1/\ln B$, the absolute value of the derivative $f'(x) = 1/(x \ln B)$ is at most 1, and thus for $x, x' \geq 1/\ln B$,

$$||\log_B(x \ln B) - \log_B(x' \ln B)||_1 \leq 1 \cdot ||x - x'||_1.$$

Since the mapping $x_i \mapsto \max\{\log_B(x_i \ln B), 0\}$ is constant for $x_i \leq 1/\ln B$, we conclude that it is neighborhood-preserving. □

We next consider the properties of this data structure.

Lemma 6.2. *Let $\varepsilon > 0$ and $x \in \mathbb{R}_+^d$ be a k-sparse vector with each coordinate $x_i \leq \beta$. Let $\alpha = \Theta(1)$ and $s = \Theta(k)$. Let $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$ and $p = 1/(\gamma + 2)$, and set $B = (4(p - p^2))^{-\varepsilon/(4\alpha)}$. Then*

- *for all $x_i \geq 1/\ln B$, the (multiplicative) expected per-entry error is*

$$\mathbb{E}[\max\{\tilde{x}_i/x_i, x_i/\tilde{x}_i\}] = O\left(1\right) \quad ;$$

- *the evaluation time is $O(\log(\beta))$; and*
- *the space is $O((k + \log(d))\log(\beta))$ bits.*

Proof. Let $\dot{x}$ be the transformed input for Algorithm 4 and $\ddot{x}$ the estimate returned by Algorithm 5. The multiplicative estimation errors of $\tilde{x}_i/x_i$ and $x_i/\tilde{x}_i$ are bounded by $B^{|\ddot{x}_i - \dot{x}_i|}$. Let S be a simple random walk with parameter p. We can bound the estimation error by considering the last nonnegative step of S similar to the proof of Lemma 4.7. The absolute error between $\dot{x}_i\varepsilon/\alpha$ and the estimate computed in Step (3) of Algorithm 3 is at most one more than the length of longest interval in which half the bits are flipped due to rounding.

Using the definition of the expectation and the calculations from Lemma 4.9, we upper bound the expected multiplicative error as follows:

$$\begin{aligned}\mathbb{E}\left[\max\{\tilde{x}_i/x_i, x_i/\tilde{x}_i\}\right] &\leq \sum_{\tau=0}^{\infty} B^{\frac{(2\tau+1)\alpha}{\varepsilon}} \cdot 2\Pr[(\max_n : S_n \geq 0) = 2\tau] \\ &\leq (2-4p)B^{\alpha/\varepsilon} \sum_{\tau=0}^{\infty} \left(B^{2\alpha/\varepsilon} 4\left(p-p^2\right)\right)^{\tau}\end{aligned}$$

We set $B = \left(4(p-p^2)\right)^{-\varepsilon/(4\alpha)}$ such that

$$\begin{aligned}(2-4p)B^{\alpha/\varepsilon} \sum_{\tau=0}^{\infty} \left(B^{2\alpha/\varepsilon} 4\left(p-p^2\right)\right)^{\tau} &= \frac{2-4p}{(4p-4p^2)^{1/4}} \sum_{\tau=0}^{\infty} \left(\sqrt{4p-4p^2}\right)^{\tau} \\ &= \frac{2-4p}{(4p-4p^2)^{1/4} - (4p-4p^2)^{3/4}} = O(1) \ .\end{aligned}$$

With our choice of B, $\log_B(x) = O(\log(x)/\varepsilon)$ and $\log_B(\ln(B)) < 1$ holds for any choice of B. The statements about running time and space usage follow directly from Theorem 4.16 using $u = 1 + O(\log(\beta)/\varepsilon)$. □

We remark that capping $\dot{x}$ to zero means that we treat entries where $x_i \leq 1/\ln B$ as $4\alpha/(\ln(1/(4p-4p^2))\varepsilon) = \Theta(1/\varepsilon)$ for our choice of B, incurring an additive error of $\Theta(1/\varepsilon)$ for small entries. The proof above shows that the expected multiplicative error is bounded by a constant for any fixed α. If α is not fixed we can choose the value to minimize the multiplicative error. As an simplified example, we ignore hash collisions such that $p = 1/(\alpha+2)$. Then the equation for the constant above is minimized with value ≈ 4.83 when $\alpha \approx 20.26$. However, the additive error is minimized for $\alpha \approx 3.07$ where $1/\ln B \approx 26.89/\varepsilon$. Thus, the choice of α depends on the trade-off between the two kinds of error. It is worth noting that the analysis for the expected multiplicative error is not tight. Simulations similar to those in Section 9 can be used to balance the trade-off based on empirical mean error.

6.1. **Applications.** We summarize the properties of the data structure in the settings studied before:

(1) If we do not combine the data structure with the thresholding technique as described in Section 5, we instantiate the algorithm described above with $\beta = u$. The running time is $O(\log u)$ and the data structure uses $O((k + \log(d)) \log u)$ bits.
(2) When combined with the thresholding technique with threshold $O(\log(d)/\varepsilon)$ for pure differential privacy or $O(\log(1/\delta)/\varepsilon)$ for (ε, δ)-differential privacy, we may set β to these values, respectively. This results in running time $O(\log(\log(d)/\varepsilon))$ and $O(\log(\log(1/\delta)/\varepsilon))$ with a space usage of $O((k + \log(d)) \log(\log(d)/\varepsilon))$ and $O((k + \log(d)) \log(\log(1/\delta)/\varepsilon))$ bits, respectively, not accounting for the space needed for the threshold data structure.

Algorithm 10: Histogram-Projection

Parameters : $\varepsilon > 0$, and $\beta, s \in \mathbb{N}$.
Input : k-sparse histogram $x \in \mathbb{N}^d$ where $s > 2k$. Sequence of universal hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_\beta)$.
Output : ε-differentially private representation of x.

(1) Construct $z \in \{0,1\}^{s \times \beta}$ by hashing the unary representations of x such that:

$$z_{a,b} = \begin{cases} 1, & \exists i : \ b \leq x_i \text{ and } h_b(i) = a \\ 0, & \text{otherwise} \end{cases}$$

(2) Apply randomized response to each bit of z such that $\tilde{z}_{a,b} = \text{RandResponse}\left(z_{a,b}, 1/(e^\varepsilon + 1)\right)$.
(3) Release h and $\tilde{z}$.

7. Improvements for Sparse Integer-valued Vectors

The algorithms we introduced so far work with real-valued vectors as input. In this section, we discuss a variation of the ALP mechanism if we restrict the input to integers. Recall that two vectors $x, x' \in \mathbb{R}_+^d$ are defined as neighboring iff $\|x - x'\|_1 \leq 1$. Under this definition neighboring vectors might disagree on several entries. As an example, two neighboring k-sparse vectors can differ by $1/2k$ in $2k$ entries. However, for the special case of histograms, that is $x, x' \in \mathbb{N}^d$, neighboring input may only disagree on one entry. We can utilize this to design a version of the ALP mechanism with improved accuracy for some values of ε. Algorithm 10 shows a projection algorithm for histograms. It is similar to Algorithm 4 without the scaling and rounding steps, and with a flip probability in randomized response that depends on ε.

Lemma 7.1. *Algorithm 10 satisfies ε-differential privacy.*

Proof. Neighboring histograms only differ in a single entry. As such at most one bit in z is changed from replacing x with x'. For randomized response with $p = 1/(e^\varepsilon + 1)$ it holds for each $b \in \{0,1\}$ that

$$\frac{\Pr[\text{RandResponse}\,(b,p) = b]}{\Pr[\text{RandResponse}\,(b,p) = 1-b]} = \frac{1-p}{p} = e^\varepsilon \ .$$

By symmetry the mechanism is ε-differentially private.

□

A key feature of Algorithm 10 is that the parameter for randomized response is a function of ε. For comparison, in Algorithm 4 it depends on the adjustable parameter α, and ε is only used for a linear scaling. For this reason Algorithm 10 is preferred for large values of ε since the noise is significantly reduced as we show next. However, the technique used in Algorithm 4 is still preferred for small ε.

For the error analysis, we first consider the expected error when there are no hash collisions. We later include hash collisions in the analysis. We estimate the value of an entry with Algorithm 3. We assume that the true value is at most β. If this is not true there is an additional error, as shown in Lemma 4.8.

Lemma 7.2. *The expected per-entry error when using Algorithm 3 on output from Algorithms 10 for entries at most β is bounded by $8 \cdot e^\varepsilon/(e^\varepsilon - 1)^2$ if there are no hash collisions.*

Proof. By the argument used in the proof of Lemma 4.7 we can bound the error by examining simple random walks with $p = 1/(e^\varepsilon + 1)$. The expected error is at most twice the expected last nonnegative step in the random walk. By Lemma 4.6 this is $4(p - p^2)/(1 - 2p)^2$, so we can bound the expected error by

$$2 \cdot \frac{4(p-p^2)}{(1-2p)^2} = 8 \cdot \frac{\frac{e^\varepsilon}{(e^\varepsilon+1)^2}}{\frac{(e^\varepsilon-1)^2}{(e^\varepsilon+1)^2}} = \frac{8e^\varepsilon}{(e^\varepsilon-1)^2} \ .$$

□

Recall from Lemma 4.14 that the expected error of Algorithm 5 is $O(1/\varepsilon)$. Therefore, we expect the above algorithm to perform better for "large" values of ε as the expectation approaches $O(1/e^\varepsilon)$ for $\varepsilon \to \infty$. However, it performs worse for "small" values as the expectation approaches $O(1/\varepsilon^2)$ for $\varepsilon \to 0$. In Section 9 we compare the error of the two algorithms for varying values of ε.

Next we consider the effect of hash collisions on the expected error. From previous sections, we know that it is sufficient to bound the probability of hash collisions by a constant for the general ALP mechanism. That is however not sufficient for large ε as the probability used for randomized response is very low. That is, a hash collision has a bigger impact on the probability of outputting a one for larger ε.

Lemma 7.3. *The expected per-entry error when using Algorithm 3 on output from Algorithms 10 for entries at most β is bounded by*

$$\frac{4e^\varepsilon}{(e^\varepsilon-1)^2} + \frac{\frac{4e^\varepsilon+4}{1+(e^\varepsilon-1)\frac{k}{s}} - 4}{\left(\frac{e^\varepsilon+1}{1+(e^\varepsilon-1)\frac{k}{s}} - 2\right)^2}.$$

Proof. We know from the proof of Lemma 4.7 that the expected last non-negative step of a simple random walk with $p = 1/(\gamma + 2)$ is $(4\gamma + 4)/\gamma^2$. Since we use universal hash functions and store at most k ones in each column the probability of hash collisions is at most k/s. Therefore, we can bound the probability of changing a bit from zero to one by $(e^\varepsilon - 1) \cdot (k/s)/(e^\varepsilon + 1)$. By setting

$$\frac{1}{\gamma+2} = \frac{1 + (e^\varepsilon - 1)\frac{k}{s}}{e^\varepsilon + 1}$$

and solving for γ, we get

$$\gamma = \frac{e^\varepsilon + 1}{1 + (e^\varepsilon - 1)\frac{k}{s}} - 2.$$

Thoerfore, we can bound the positive error by

$$\frac{4\gamma+4}{\gamma^2} = \frac{\frac{4e^\varepsilon+4}{1+(e^\varepsilon-1)\frac{k}{s}} - 4}{\left(\frac{e^\varepsilon+1}{1+(e^\varepsilon-1)\frac{k}{s}} - 2\right)^2} \ .$$

The expected negative error is still bounded by $4e^\varepsilon/(e^\varepsilon - 1)^2$ since hash collisions have no impact. □

Algorithm 11: Max-Projection

Parameters: $\varepsilon > 0$, and $s \in \mathbb{N}$.
Input : $x \in \mathbb{R}_+^d$. A universal hash function h from domain $[d]$ to $[s]$.
Output : ε-differentially private representation of x.

(1) Initialize $y \in \mathbb{R}^s$ to the all zeroes vector.
(2) For each $i \in [s]$, set $y_i = \max_{h(j)=i} x_j$.
(3) For each $i \in [s]$, sample $\eta_i \sim \text{Lap}(1/\varepsilon)$
(4) Release h and $\tilde{y} = y + (\eta_1, \ldots, \eta_s)$.

If we apply the thresholding techniques before running Algorithm 10, the number of bits needed to store $\tilde{z}$ is $O((s \log d)/\varepsilon)$ and $O((s \log(1/\delta))/\varepsilon)$, respectively. By setting $s = \Theta(k)$ we bound the probability of hash collisions by a constant. This works decently for small values of ε, but we need to use more space for large values. When the probability of a hash collision is q, the probability of flipping a bit from zero to one is at least $q/2$. This would put a lower bound on the error for any ε. We can use some extra space to get error $O(1/\varepsilon)$ or $O(1/e^\varepsilon)$ expected error for large epsilon as summarized below. The corollary follows directly from plugging in different choices of s into Lemma 7.3.

Corollary 7.4. *Assuming that the threshold technique for ε-DP (Algorithm 6) was applied, using Algorithm 3 on output from Algorithms 10 has the following properties:*

(1) *Let $s = \Theta(k\varepsilon)$ and $\varepsilon > 1$. Then the expected error for $\varepsilon \to \infty$ is $O(1/\varepsilon)$ and $\tilde{z}$ is stored in $O(k \log d)$ bits.*
(2) *Let $s = \Theta(ke^\varepsilon)$ and $\varepsilon > 1$. Then the expected error for $\varepsilon \to \infty$ is $O(1/e^\varepsilon)$ and $\tilde{z}$ is stored in $O(e^\varepsilon k \log(d)/\varepsilon)$ bits.*

The results extend naturally to the case of (ε, δ)-DP using Algorithm 9.

8. Constant Access Time with Optimal Expected Error

In this section, we discuss a data structure that achieves access time $O(1)$ with expected per-entry error $O(1/\varepsilon)$, improving on the mechanisms discussed in previous sections. As a downside, the error bound is only in expectation and the data structure does not have strong tail bounds as compared to the ALP mechanism, cf. Lemma 4.11. The data structure is inspired by the Count-Min sketch [CM05].

8.1. **The data structure.** Algorithm 11 shows the *projection algorithm* that returns a differentially private data structure that can be used for estimation. Given a k-sparse vector $x \in \mathbb{R}_+^d$ and a random hash function h mapping from $[d]$ to $[s]$, the algorithm returns a vector $\tilde{y} \in \mathbb{R}^s$ for a parameter s to be chosen later. The idea is that, before noise, each coordinate y_i stores the maximum entry in x for all coordinates of x that are mapped to i by h. We use the Laplace mechanism on y to release $\tilde{y}$ as the differential private version of x. Given a coordinate $i \in [d]$, we *estimate* x_i as $\tilde{y}_{h(i)}$.

Lemma 8.1. *Algorithm 11 satisfies ε-differential privacy.*

Proof. Let x and x' be such that $||x - x'||_1 \leq 1$, and define y and y' as in Line (2) of Algorithm 11. Each coordinate i such that x_i and x'_i differ can contribute a change of not

more than $|x_i - x_i'|$ to $||y - y'||_1$. Thus, $||y - y'||_1 \leq 1$. Adding Laplace noise with scale $1/\varepsilon$ guarantees ε-differential privacy. □

Lemma 8.2. *Given $x \in \mathbb{R}_+^d$ and $\varepsilon > 0$, let $h, \tilde{y}$ be the output of Algorithm 11 with $s = \Omega(\varepsilon||x||_1)$. For each $i \in [d]$, $\mathbb{E}[|x_i - \tilde{y}_{h(i)}|] = O(1/\varepsilon)$. The evaluation of a single coordinate takes time $O(1)$.*

Proof. The running time statement follows because the algorithm evaluates a single hash function value.

Let J be the set of coordinates of the non-zero entries in x. We can use that the hash function is universal to bound the expected difference between x_i and $y_{h(i)}$ by

$$\begin{aligned}\mathbb{E}\left[|x_i - y_{h(i)}|\right] &\leq \sum_{j \in J} \Pr(h(i) = h(j)) \cdot \max(x_j - x_i, 0) \\ &\leq \frac{1}{s} \sum_{j \in J} x_j = \frac{\|x\|_1}{s}.\end{aligned}$$

The expected error from the Laplace noise is $O(1/\varepsilon)$ by Proposition 2.5. For $s = \Omega(\varepsilon||x||_1)$, the expected error is $O(1/\varepsilon)$ for zero entries. Since each non-zero entry x_j with $h(i) = h(j)$ potentially contributes to $y_{h(i)}$, the expected error is only smaller for non-zeroes. □

Corollary 8.3. *If $\varepsilon > 0$ and $x \in \mathbb{R}_+^d$, then*

1. *If $||x||_1 = n$, the data structure described above uses space $s = O(\varepsilon n)$ words, guarantees constant access time, and has expected error $O(1/\varepsilon)$;*
2. *If x is k-sparse, then the data structure uses $O(k \log d)$ words, has constant access time, and has expected error $O(1/\varepsilon)$.*

Proof. If $||x||_1 = n$ is known, we can use $s = \Theta(n\varepsilon)$ as the size of the table. If the vector is k-sparse, we use the thresholding technique (Algorithm 6) with threshold $O(\log(d)/\varepsilon)$ and store the entries below the threshold using Algorithm 11. Restricting on the elements below the threshold, we know that $||x||_1 = O(k \log(d)/\varepsilon)$, and the result follows. □

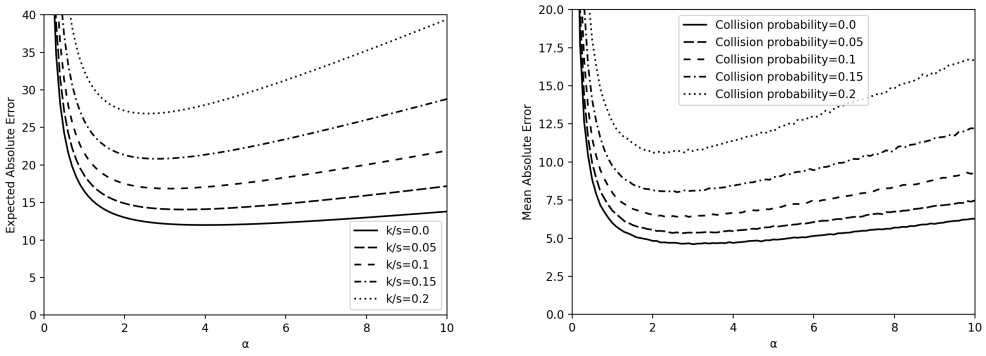
This approach guarantees constant access time with expected error $O(1/\varepsilon)$, but does not guarantee good tail bounds similar to the ALP mechanism and the Laplace mechanism. Let us focus on the case that we use $s = O(k \log d)$ for a k-sparse vector $x \in \mathbb{R}_+^d$. The probability that one of the $d - k$ zero entries collides with a non-zero is $O(1/\log d)$. All of the non-zero entries can be as large as $O(\log(d)/\varepsilon)$. Thus, we expect $(d - k)/\log d$ zero entries to have error $O(\log(d)/\varepsilon)$.

9. Experiments

In this section, we discuss the per-entry error of ALP1-Estimator (Algorithm 3) in practice. Let $\gamma = (\alpha + 2)/(1 + \alpha k/s) - 2$. By Lemma 4.4 and 4.7 the expected per-entry error of ALP1-Estimator is upper bounded by:

$$\mathbb{E}[|x_i - \tilde{x}_i|] \leq \left(\frac{1}{2} + \frac{4\alpha + 4}{\alpha^2} + \frac{4\gamma + 4}{\gamma^2}\right) \cdot \alpha \ .$$

Figure 3(a) shows the upper bound for varying values of k/s and α. Recall that k/s is a bound on the probability of a hash collision. We see that the effect of hash collisions on



(a) Upper bound on expected per-entry error (b) Observed mean per-entry error

FIGURE 3. Theoretical expected per-entry error and experiment results. Note that the y-axes for the plots use different scales.

the error increases for large values of α, as each bit in the embedding is more significant. We discuss how the upper bound compares to practice next.

Experimental Setup. We designed experiments to evaluate the effect of the adjustable parameters α and s on the expected per-entry error of ALP1-Estimator. The experiments were performed on artificial data. For our setup, we set β equal to 5000 and chose a true value x_i uniformly at random in the interval $[0; \beta]$. We run only on artificial data, as uniform data does not benefit the algorithm, and we can easily simulate worst-case conditions for hash collisions. We simulate running the ALP1-Projection algorithm by computing y_i, simulating hash collisions, and applying randomized response. The probability for hash collisions is fixed in each experiment, and the same probability is used for all bits. This simulates worst-case input in which all other non-zero entries have a true value of at least β. We increment α by steps of 0.1 in the interval $[0.1, \ldots, 10]$ and the probability of a hash collision by 0.05 in the interval $[0, \ldots, 0.2]$. The probability of 0 serves only as a baseline, as it is not achievable in practice for $k > 1$. The experiment was repeated 10^5 times for every data point.

Figure 3(b) shows plots of the mean absolute error of the experiments. As α is increased, the error drops off at first and slowly climbs. The estimates of y_i are more accurate for large values of α. However, any inaccuracy is more significant, as $\tilde{y}_i$ is scaled back by a larger value. The error from the random rounding step also increases with α. The plots of the upper bound and observed error follow similar trajectories. However, the upper bound is approximately twice as large for most parameters.

Fixed Parameters. The experiments show how different values of α and s affect the expected per-entry error. However, the parameters also determine constant factors for space usage and access time. The space requirements scale linearly in s/α, and the access time is inversely proportional to α. As such, the optimal parameter choice depends on the use case due to space, access time, and error trade-offs.

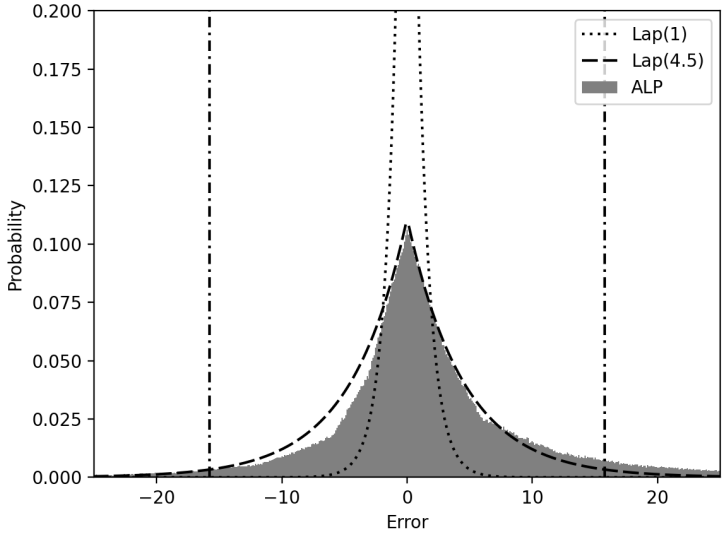


FIGURE 4. Error distribution of Algorithm 3.
($\varepsilon = 1$, $\alpha = 3$, and collision probability $= 0.1$)

To evaluate the error distribution of the ALP1-Estimator algorithm we fixed the parameters of an experiment. We set $\alpha = 3$ and the hash collision probability to 0.1. We repeated the experiment 10^6 times.

The error distribution is shown in Figure 4. The mean absolute error of the experiment is 6.4 and the standard deviation is 11. Plugging in the parameters in Lemma 4.11, with probability at least 0.9 the error is at most

$$|x_i - \tilde{x}_i| < 3 + \frac{6 \log\left(\frac{5}{0.12\sqrt{\pi}}\right)}{\log\left(\frac{25}{19.24}\right)} \approx 75.33 \ .$$

The error of the observed 90th percentile is 15.78, which is shown in Figure 4 using vertical lines. Again, this shows that the upper bounds are pessimistic.

For comparison, the plots include the Laplace distribution with scale parameters 1 and 4.5. Note that the Laplace distribution with parameter 1 is optimal for the privacy budget. The standard deviation of the distribution with scale 4.5 is 6.36 and the mean absolute error is similar to the ALP mechanism.

The distribution is slightly off-center, and the mean error is 2.33. This is expected due to hash collisions. The effect of hash collisions is also apparent for the largest observed errors. The lowest observed error was -114, while the highest was 274. There is a clear trade-off between space usage and per-entry error. We reran the experiment with hash collision probability 0.01 using the same value for α. The error improved for all the metrics mentioned above. The mean absolute error is 4.8, the standard deviation is 7.8, the mean error is 0.18, the 90th percentile is 11.5, and the largest observed error is 147.

Histograms. Next we compare the expected per-entry error of ALP-Projection (Algorithm 4) with the variant designed for histograms introduced in Section 7. As discussed in

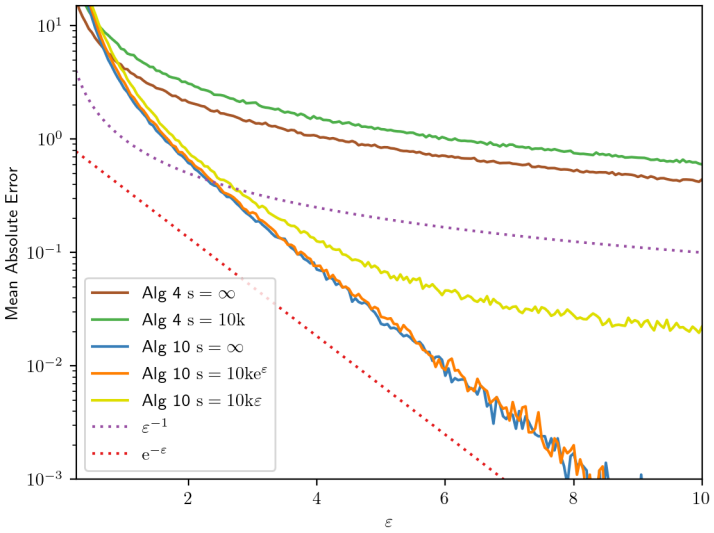


FIGURE 5. Mean per-entry error of Algorithms 4 and 10

the section, Algorithm 10 is more sensitive to ε than Algorithm 4. The experiment in this section compares the effect of changing ε on both algorithms. In particular, the experiments are designed to examine when Algorithm 10 is preferred.

For our setup, we again set β to 5000. We increment ε by 0.05 in the interval $[0.25, \ldots, 10]$. For algorithm 10 we choose an integer uniformly in $[0, \ldots, \beta]$. We set α equal to 3 based on the previous experiments. Recall that Algorithm 4 introduces error in the scaling step if x_i is not a multiple of α/ε. For this reason we uniformly select a multiple of α/ε in the interval $[0; \beta]$. The experiment was repeated 10^5 times for each data point.

We ran both simulations with no probability of hash collisions as a baseline. We denote this as $s = \infty$. We ran Algorithm 4 with a probability of 0.1 for hash collisions. We denote this as $s = 10k$ in the legend. For Algorithm 10 we ran the experiment with probabilities of hash collisions of $0.1/e^{\varepsilon}$ and $0.1/\varepsilon$. However, we use 0.1 for $\varepsilon < 1$. The result of the experiment is shown in Figure 5. The y-axis has a logarithmic scale and the functions $1/\varepsilon$ and $1/e^{\varepsilon}$ are included for comparison.

The plots show that, as expected, the preferred algorithm depends on the value of ε. We see that Algorithm 10 is preferred when ε is approximately 0.7 and above. Also as expected, the error of Algorithm 10 is $O(1/e^{\varepsilon})$ and $O(1/\varepsilon)$ for large epsilon if the probabilities of hash collisions is $\Theta(1/e^{\varepsilon})$ and $\Theta(1/\varepsilon)$, respectively. However, it still outperforms Algorithm 4 for large ε by more than an order of magnitude.

Note that Algorithm 4 is much preferred for small ε even though it is not as clearly visible from the Figure. As discussed in Section 7, Algorithm 10 uses more space and the expected error scales roughly as $O(1/\varepsilon^2)$ for $\varepsilon < 1$. We ran the experiment with $\varepsilon = 0.05$ and Algorithm 4 outperformed the mean error of Algorithm 10 by an order of magnitude.

10. Suggestions to Practitioners

The ALP mechanism introduced in this paper combines the best of three worlds: It has low error similar to the Laplace mechanism, produces compact representations using asymptotically optimal space, and has an access time that scales only with $O(\log d)$.

In an application that wants to make use of differentially private histograms/vectors, one first has to get an overview of the assumed properties of the data before making a choice on which approach to use. If d is small or the data are assumed to be dense, the Laplace mechanism will offer the best performance. If the data are sparse and the dimension d is large, the analyst must know which error guarantee she wishes to achieve, and which access time is feasible in the setting where the application is deployed. If a larger error is acceptable for "small" entries or access time is crucial, just applying the thresholding technique [KKMN09, CPST12] is the better choice. Otherwise, if small error is paramount or an access time of $O(\log d)$ is sufficient, the ALP mechanism will provide the best solution.

Variants. We assume in this paper that k is a known bound on the sparsity of the input data. However, in some applications the value of k itself is private. Here we briefly discuss approaches in such settings. We use the value of k to select the size of the embedding, such that the probability of hash collisions is sufficiently small. When k is not known, we can still bound the probability of hash collisions.

If the input is a *histogram* the sparsity differs by at most 1 for neighboring datasets. As such we can use a fraction of the privacy budget to estimate the sparsity. Note that this is not possible for real-valued vectors, as the difference in sparsity can be as large as d for neighboring datasets.

If $\|x\|_1 = n$ is known, then we have $\|\hat{x}\|_1 = n\varepsilon$ for the scaled input. We can bound the probability of hash collisions by a constant when the size of the embedding is $\Theta(n\varepsilon)$ bits. We also have $\|x\|_\infty \leq n$ and as such we can set $u = n$ if no better bound is known. If $\|x\|_1$ is unknown, we can estimate it using a fraction of the privacy budget. Note that the space differs from the k-sparse setting, and remains $\Theta(n\varepsilon)$ bits when applying the thresholding techniques.

In both cases the estimate affects space and error of our mechanism. We discuss estimating k here, but the same principle applies to estimating $\|x\|_1$. Let $\tilde{k}$ be the estimate of k used to set s such that $s = \Theta(\tilde{k})$. It is clear that the space requirement now scales with $\tilde{k}$ instead of k. However, the error guarantees of our mechanism depend on k/s. We can bound this by a constant when k is known, but doing so might introduce large error if $\tilde{k}/k$ is small. This is only likely to happen if the true value of k is small. If a small error is more important than space, we can estimate $\tilde{k}$ such that $k \leq \tilde{k}$ with high probability. For example, we can set $\tilde{k} = k + \mathrm{Lap}(1/\varepsilon) + \ln(d/2)/\varepsilon$ such that $k \geq \tilde{k}$ only happens with probability at most $1/d$.

In Section 2 we gave a simple reduction from real-valued data to nonnegative, real-valued data with increased error. It is possible to extend our mechanism to negative values by instead paying an increase in access time. The thresholding techniques in Algorithms 6 and 9 are easily extended to real values by releasing noisy entries with a large absolute value. However, Algorithm 9 only releases entries whose true and noisy values have the same sign to preserve the privacy guarantees. We extend Algorithm 2 by using twice as many columns to store z. The last m columns store nonnegative values as before. The first m columns store

the negative values. The bits are set to 1 by default and changed to 0 to encode negative values. Algorithm 3 is unchanged, but an offset is used for the returned value.

An implementation of a variant of the ALP mechanism is available as part of the open source project OpenDP (https://opendp.org/) in the repository https://github.com/opendp/opendp.

11. Open problems

The main open problem that we leave is if it is possible to achieve similar space and error with constant time access. In Section 8 we demonstrated how to achieve optimal *expected* error with constant time access and space within a logarithmic factor of optimal. However, this method does not have strong tail bounds on the error.

Acknowledgments. We thank the anonymous reviewers of the TPDP'21 workshop, and the ACM CCS conference, and the *Journal of Privacy and Confidentiality* for their detailed suggestions that helped improve the paper. We thank Jakub Tětek for discussions that led to the results mentioned in Section 6. Christian Janos Lebeda and Rasmus Pagh are affiliated with Basic Algorithms Research Copenhagen (BARC), supported by the VILLUM Foundation grant 16582.

References

[Alm02] Sven Erick Alm. Simple random walk. *Unpublished manuscript*, 2002. URL: http://www2.math.uu.se/~sea/kurser/stokprocmn1/slumpvandring_eng.pdf.

[BNS19] Mark Bun, Kobbi Nissim, and Uri Stemmer. Simultaneous private learning of multiple concepts. *J. Mach. Learn. Res.*, 20:94:1–94:34, 2019. URL: http://jmlr.org/papers/v20/18-549.html.

[BV19] Victor Balcer and Salil P. Vadhan. Differential privacy on finite computers. *J. Priv. Confidentiality*, 9(2), 2019. https://doi.org/10.29012/jpc.679.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. https://doi.org/10.1016/j.jalgor.2003.12.001.

[CPST12] Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. Differentially private summaries for sparse data. In *ICDT*, pages 299–311. ACM, 2012, pages 299–311. https://doi.org/10.1145/2274576.2274608.

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. https://doi.org/10.1016/0022-0000(79)90044-8.

[DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997. https://doi.org/10.1006/jagm.1997.0873.

[DMNS16] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7(3):17–51, 2016. https://doi.org/10.29012/jpc.v7i3.405.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. https://doi.org/10.1561/0400000042.

[Elk13] Noam D. Elkies. Upper limit on the central binomial coefficient, 2013. [Online; accessed 15-September-2021]. URL: https://mathoverflow.net/questions/133732/upper-limit-on-the-central-binomial-coefficient.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed.* Addison-Wesley, 1994. URL: https://www-cs-faculty.stanford.edu/%7Eknuth/gkp.html.

[Hag98] Torben Hagerup. Sorting and searching on the word RAM. In *STACS*, volume 1373 of *Lecture Notes in Computer Science*, pages 366–398. Springer, 1998, 1373:366–398. https://doi.org/10.1007/BFb0028575.

[HT10] Moritz Hardt and Kunal Talwar. On the geometry of differential privacy. In *STOC*, pages 705–714. ACM, 2010, pages 705–714. https://doi.org/10.1145/1806689.1806786.

[KHP15] Fragkiskos Koufogiannis, Shuo Han, and George J. Pappas. Optimality of the laplace mechanism in differential privacy. *CoRR*, abs/1504.00065, 2015. https://doi.org/10.48550/ARXIV.1504.00065.

[KKMN09] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180. ACM, 2009, pages 171–180. https://doi.org/10.1145/1526709.1526733.

[War65] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. https://doi.org/10.1080/01621459.1965.10480775.

Appendix A. Closed-form proof of Lemma 4.6

Here we provide a closed-form expression used in the proof of Lemma 4.6.

In the proof, we will make use of general binomial coefficient([GKP94, Equation 5.1]):

$$\binom{r}{k} = \frac{r(r-1)\dots(r-k+2)(r-k+1)}{k!} \ ,$$

and the binomial theorem ([GKP94, Equation 5.12]):

$$(1+z)^r = \sum_{k=0}^{\infty} \binom{r}{k} (z)^k \ .$$

Starting from an infinite series with $z < 1/4$, we simplify as follows:

$$\begin{aligned}
\sum_{k=0}^{\infty} k\binom{2k}{k}(z)^k &= \sum_{k=1}^{\infty} k\frac{(2k)!}{k!k!}z^k \\
&= \sum_{k=1}^{\infty} k\frac{k(k-\frac{1}{2})(k-1)\dots\left(\frac{3}{2}\right)1\left(\frac{1}{2}\right)}{k!k!}2^{2k}z^k \\
&= \sum_{k=1}^{\infty} \frac{(k-\frac{1}{2})(k-\frac{3}{2})\dots\left(\frac{5}{2}\right)\left(\frac{3}{2}\right)\left(\frac{1}{2}\right)}{(k-1)!}(4z)^k \\
&= 2z\sum_{k=1}^{\infty} \frac{\left(-\frac{3}{2}\right)\left(-\frac{5}{2}\right)\dots(-k+\frac{3}{2})(-k+\frac{1}{2})}{(k-1)!}(-4z)^{k-1} \\
&= 2z\sum_{k=1}^{\infty} \binom{-\frac{3}{2}}{k-1}(-4z)^{k-1} \\
&= 2z\sum_{k=0}^{\infty} \binom{-\frac{3}{2}}{k}(-4z)^{k} \\
&= \frac{2z}{(1-4z)^{3/2}} \ .
\end{aligned}$$

Finally, let $z = p - p^2$ for $p < 1/2$. This gives us the closed-form expression:

$$\begin{aligned}
\sum_{k=0}^{\infty} k\binom{2k}{k}(p-p^2)^k &= \frac{2(p-p^2)}{(1-4(p-p^2))^{3/2}} \\
&= \frac{2(p-p^2)}{(1-2p)^3} \ .
\end{aligned}$$