



# Space-Optimal Profile Estimation in Data Streams with Applications to Symmetric Functions

Justin Y. Chen  

Massachusetts Institute of Technology, Cambridge, MA, USA

Piotr Indyk  

Massachusetts Institute of Technology, Cambridge, MA, USA

David P. Woodruff  

Carnegie Mellon University, Pittsburgh, PA, USA

---

## Abstract

We revisit the problem of estimating the profile (also known as the rarity) in the data stream model. Given a sequence of  $m$  elements from a universe of size  $n$ , its profile is a vector  $\phi$  whose  $i$ -th entry  $\phi_i$  represents the number of distinct elements that appear in the stream exactly  $i$  times. A classic paper by Datar and Muthukrishnan from 2002 gave an algorithm which estimates any entry  $\phi_i$  up to an additive error of  $\pm\epsilon D$  using  $O(1/\epsilon^2(\log n + \log m))$  bits of space, where  $D$  is the number of distinct elements in the stream.

In this paper, we considerably improve on this result by designing an algorithm which simultaneously estimates many coordinates of the profile vector  $\phi$  up to small overall error. We give an algorithm which, with constant probability, produces an estimated profile  $\hat{\phi}$  with the following guarantees in terms of space and estimation error:

(a) For any constant  $\tau$ , with  $O(1/\epsilon^2 + \log n)$  bits of space,  $\sum_{i=1}^{\tau} |\phi_i - \hat{\phi}_i| \leq \epsilon D$ .

(b) With  $O(1/\epsilon^2 \log(1/\epsilon) + \log n + \log \log m)$  bits of space,  $\sum_{i=1}^m |\phi_i - \hat{\phi}_i| \leq \epsilon m$ .

In addition to bounding the error across multiple coordinates, our space bounds separate the terms that depend on  $1/\epsilon$  and those that depend on  $n$  and  $m$ . We prove matching lower bounds on space in both regimes.

Application of our profile estimation algorithm gives estimates within error  $\pm\epsilon D$  of several symmetric functions of frequencies in  $O(1/\epsilon^2 + \log n)$  bits. This generalizes space-optimal algorithms for the distinct elements problems to other problems including estimating the Huber and Tukey losses as well as frequency cap statistics.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Streaming, sublinear and near linear time algorithms; Theory of computation  $\rightarrow$  Sketching and sampling

**Keywords and phrases** Streaming and Sketching Algorithms, Sublinear Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2024.32

**Related Version** *Full Version:* <https://arxiv.org/abs/2311.17868>

**Funding** *Justin Y. Chen:* Supported by an NSF Graduate Research Fellowship under Grant No. 174530.

*Piotr Indyk:* Supported by the NSF TRIPODS program (award DMS-2022448) and the Simons Investigator Award.

*David P. Woodruff:* Supported in part by a Simons Investigator Award.

## 1 Introduction

Estimating basic statistics of a data set, such as the number of times each element occurs or the number of distinct elements are fundamental problems in data stream algorithms. In this paper, we focus on the related problem of estimating the number of elements that occur a given number of times. Formally, given a stream  $\mathbf{x} = x_1, \dots, x_m$  of  $m$  elements



© Justin Y. Chen, Piotr Indyk, and David P. Woodruff;  
licensed under Creative Commons License CC-BY 4.0

15th Innovations in Theoretical Computer Science Conference (ITCS 2024).

Editor: Venkatesan Guruswami; Article No. 32; pp. 32:1–32:22

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

from  $[n] = \{1, 2, \dots, n\}$ , we define its *profile* to be the frequency of frequencies vector  $\phi \in \{0, \dots, n\}^m$  where  $\phi_i = |\{j \in [n] : |\{k \in [m] : x_k = j\}| = i\}|$  is the number of distinct elements in  $\mathbf{x}$  which appear exactly  $i$  times; such elements are often referred to as “ $i$ -rare” and were first studied in a classic paper of Datar and Muthukrishnan [15].

Any *symmetric* function of the frequencies of the elements in a stream (such a function is invariant to relabeling of the domain elements) can be written as a function of the profile. Therefore, algorithms for profile estimation in data streams can be used to estimate quantities such as the number of distinct elements [19], frequency moments [2], capped statistics of the stream [11], or the objective function of  $M$ -estimators such as the Huber or Tukey objective [24]. In this paper, we develop an algorithm for estimating the profile of a data set specified by an insertion-only stream, where elements are inserted but not deleted. As an application, this algorithm improves upon the space complexity of estimating several symmetric functions of frequencies.

The profile (also referred to as the fingerprint, histogram, histogram of histograms, pattern, prevalence, or collision statistics) of a data set is a natural representation of the distribution of elements and has been studied extensively from both computational and statistical perspectives. Streaming algorithms that estimate the number of  $i$ -rare elements have been used for computing degree distributions in large graphs [7], detecting malicious IP traffic in a network [27], estimating the number of times users have been exposed to the same ad [20], counting the number of  $k$ -mers in genetic sequences with a given abundance value for fast  $k$ -mer size selection [10], and for applications in databases [14]. In practice, estimating the profile is a very popular sketching problem solved by users of Apache DataSketches, a popular open-source sketching library [16]<sup>1</sup>.

The study of the problem in the context of streaming algorithms dates back to the work of Datar and Muthukrishnan [15]. They show how to estimate the ratio of  $i$ -rare elements in the stream to the total number of distinct elements  $D$  in the stream, i.e., the fraction  $\phi_i/D$ . The algorithm is simple and elegant: it collects a random sample  $j_1 \dots j_s$  of  $s$  elements from the stream, where each  $j_t$  is chosen uniformly at random from the set of distinct elements appearing in the stream. For each element  $j_t$ , it calculates the element’s frequency (the number of times  $j_t$  appears in the stream) and returns the fraction of  $j_t$ ’s with frequency exactly  $i$ . The algorithm can be implemented in one pass using  $O(s(\log n + \log m))$  bits of storage, and the authors show a trade-off between the quality of approximation and the sample size  $s$ . Specifically, to approximate  $\phi_i/D$  up to  $\pm\epsilon$  with constant probability, it suffices that  $s = \Theta(1/\epsilon^2)^2$ . This translates into an  $O(1/\epsilon^2(\log n + \log m))$  space streaming algorithm which, given a particular  $i$ , finds an estimate  $\hat{\phi}_i$  such that

$$|\phi_i - \hat{\phi}_i| \leq \epsilon D. \tag{1}$$

Other works have studied rarity estimation streaming algorithms in the context of the sliding window model [6], time and space efficient algorithms [18] and privacy [17]. The “layering” technique of Indyk and Woodruff [23] essentially estimates the number of items whose frequencies are *approximately* equal to a given value. A pertinent line of work studies algorithms for estimating general “concave sublinear” frequency statistics which depend on the rarities of low frequency elements [12, 13]. These papers provide succinct sketches for estimating several symmetric functions of frequencies but either do not account for the space

<sup>1</sup> See this example of estimating the number of estimating the distribution of how many times users visit a website in a month <https://datasketches.apache.org/docs/Tuple/TupleEngagementExample.html>.

<sup>2</sup> The original paper provides a more refined bound with a mix of additive and multiplicative errors, see Lemma 2 in [15]. We provide a single-parameter bound for the sake of simplicity.

used to store hash functions or do not focus on bit complexity and require two passes over the data. When the space used for randomness is attributed to the algorithm, our results give improved space bounds compared to prior work for estimating several statistics that fall into this framework (see Section 1.2).

## 1.1 Results for Profile Estimation

We focus on estimating several coordinates of the profile vector  $\phi$  simultaneously with small *total* error. In many applications, it is useful to calculate more than one rarity as the true object of interest is the distribution of frequencies. In addition, estimating several coordinates of the profile has direct applications to estimating several symmetric functions of frequencies (see Section 1.2). Our main result is a streaming algorithm which, with constant probability, achieves the following guarantees (Theorems 1 and 2):

- For any  $\tau = O(1)$ , using space  $O(1/\epsilon^2 + \log n)$ , the algorithm returns a  $\hat{\phi}$  such that

$$\sum_{i=1}^{\tau} |\phi_i - \hat{\phi}_i| \leq \epsilon D. \quad (2)$$

- Using space  $O(1/\epsilon^2 \log(1/\epsilon) + \log n + \log \log m)$ , the algorithm returns a  $\hat{\phi}$  such that

$$\sum_{i=1}^m |\phi_i - \hat{\phi}_i| \leq \epsilon m. \quad (3)$$

Both results use less space than the algorithm of Datar and Muthukrishnan while bounding error across several coordinates of the profile rather than for a single rarity. A brief remark comparing the two guarantees: as  $D \leq m$ , the error in Equation (2) is smaller than that of Equation (3) but at the cost of only providing a guarantee for estimating the profile over constant frequencies.<sup>3</sup> On the other hand, by Equation (3) we can estimate the entire profile in small space up to error  $\pm \epsilon m$ . Note that in common settings where the average frequency of the elements in the stream is small,  $\epsilon m$  and  $\epsilon D$  are of similar size. However, in general, estimating the entire profile up to additive error in terms of  $D$  is hard. In fact, producing an estimate such that  $\|\phi - \hat{\phi}\|_1 \leq D/2$  would require  $\Omega(D \log m)$  bits of space.<sup>4</sup>

The first step in our analysis is to show that  $s = O(1/\epsilon^2 \log(1/\epsilon))$  samples suffice to achieve the guarantees of Equation (3) using the empirical estimation method of Datar and Muthukrishnan ( $s = O(1/\epsilon^2)$  suffices for Equation (2) from the original paper). However, this naively requires  $s(\log n + \log m)$  space to store the identities and counts of each sample.

Our main result reduces the space required to  $O(s)$  bits. The algorithm is obtained by compressing the identity and count information of each sample into  $O(1)$  bits on average while retaining its statistical power for estimating the profile. The algorithm is also time efficient: it takes  $O(\log(1/\epsilon) + \log \log n)$  expected amortized time to process each stream update and  $\text{poly}(1/\epsilon)$  time to produce the final profile estimate.

<sup>3</sup> Note that in this regime, to estimate the entire profile up to the  $\tau$ th coordinate, it is sufficient to have an algorithm for estimating a single  $i$ -rarity for constant  $i$  and making a constant number of copies of this algorithm for each  $i \in \{1, \dots, \tau\}$ . Interestingly, our algorithm internally produces an estimate of the entire profile up to  $i$  even if the goal is only to estimate the single  $i$ -rarity.

<sup>4</sup> Let  $S_N^D$  be the set of all  $D$ -sparse binary vectors of length  $N$ , and let  $C \subset S_N^D$  be its subset such that any pair of distinct  $c, c' \in C$  have  $L_1$  distance greater than  $D$ . Standard probabilistic arguments show that there exists such a set  $C$  of size  $\exp(\Omega(D \log(N/D)))$ . Observe that for any  $c \in C$ , we can generate a stream with profile equal to  $c$ , by creating, for each nonzero  $c_i$ , a distinct element appearing  $i$  times. Given such a stream, the assumed algorithm returns  $\hat{\phi}$  with  $L_1$  distance at most  $D/2$  to  $c$ , which makes it possible to uniquely recover  $c$ . By the pigeonhole principle, the algorithm must use space at least  $\Omega(\log |C|) = \Omega(D \log(N/D))$ . Since the length of the generated stream  $m$  is at most  $ND$ , if we pick  $N > D^2$ , we obtain the desired bound.

To achieve these results, we use *two new techniques*:

1. To compress the identities of the stream elements, we hash the sampled elements to a domain of size  $O(s)$ , allowing collisions between sampled elements in a similar manner to Bloom filters or CountMin sketches. Our key contribution is to show that, under the parameters of our algorithm, the empirical profile of the sampled elements can be approximately recovered from the frequencies after hashing by an iterative “inversion” procedure. To our knowledge, this inversion procedure is novel and requires a careful analysis of the hashing procedure as well as an application of the “Poissonization” trick more commonly used in distribution testing.
2. To use small space, we need to be able to make efficient use of randomness for our hash functions. A key statistic used in our analysis is the number of buckets in the hash table with frequency  $i$  for  $i \in [m]$ . Our analysis requires  $O(1)$ -wise independence of the associated random variables, however, this does not simply follow from using an  $O(1)$ -wise independent hash family. On the other hand, Nisan’s generator can be used in a black-box fashion [22], but this would blow up the space bound by a logarithmic factor. Instead, we apply Nisan’s generator to a subroutine of the streaming algorithm to ensure that  $O(1)$ -wise independence holds for the pertinent random variables. To our knowledge, this is a novel technique and one that seems quite versatile: since its introduction in an earlier version of this paper, it has been already used for other streaming problems [25].

We complement our algorithmic results with the following lower bounds which show that we achieve the *optimal* dependence on the error parameter  $\epsilon$  (Theorems 14 and 13).

- Any one-pass algorithm satisfying Equation (2) with constant probability must use at least  $\Omega(1/\epsilon^2)$  bits of space.
- Any one-pass algorithm satisfying Equation (3) with constant probability must use at least  $\Omega(1/\epsilon^2 \log(1/\epsilon))$  bits of space.

To the best of our knowledge, the latter is a rare example of a natural streaming problem where the optimal dependence of the space bound on the accuracy parameter  $\epsilon$  is not of the form  $1/\epsilon^a$  for some integer exponent  $a \geq 1$ .

## 1.2 Applications to Symmetric Functions

By itself, the profile is a useful statistic of the stream, but it is also important in that any symmetric (invariant to relabeling) function of frequencies can be written as a function of the profile. Therefore, the guarantees of the algorithm given in Equation (2) and Equation (3) can be leveraged in a black-box way to give streaming algorithms for estimating a variety of symmetric functions of the frequencies of the stream. We give several illustrative examples where we can estimate functions in essentially the same space required to estimate the number of distinct elements. In what follows, consider constant  $\tau$ .

- **Distinct elements with frequency at most or at least  $\tau$ .** The number of distinct elements with frequency at most  $\tau$  is the sum of the first  $\tau$  coordinates of the profile and can be calculated up to  $\pm\epsilon D$  in  $O(1/\epsilon^2 + \log n)$  space using our algorithm. The number of distinct elements with frequency at least  $\tau$  can be calculated by subtracting those with frequency at most  $\tau - 1$  from the total number of distinct elements which can also be approximated in  $O(1/\epsilon^2 + \log n)$  space [26].
- **Mass of elements with frequency at most or at least  $\tau$ .** The mass of the distinct elements with frequency at most  $\tau$  can be expressed as  $\sum_{i=1}^{\tau} \phi_i \cdot i$  which can be calculated in space  $\pm\epsilon D$  in  $O(1/\epsilon^2 + \log n)$  space using our algorithm. Subtracting from the total mass of the stream which can be approximated up to  $\pm\epsilon m$  in space  $O(\log \log m + \log(1/\epsilon))$  with a Morris counter [28] yields the mass of elements with frequency at least  $\tau$ .

- **Capped (or saturated) statistics [11].** For a given parameter  $\tau$ , the corresponding capped statistic of the stream is

$$\sum_{i=1}^{\tau} \phi_i \cdot i + \sum_{i=\tau+1}^m \phi_i,$$

a generalization of counting the stream length and counting the number of distinct elements. This can be calculated using the two quantities above up to error  $\pm \epsilon D$  in  $O(1/\epsilon^2 + \log n)$  space.

- **Tukey objective [30].** For a given parameter  $\tau$ , the Tukey objective is

$$\sum_{i=1}^{\tau} \phi_i \cdot \frac{\tau^2}{6} \left(1 - (1 - i^2/\tau^2)^3\right) + \sum_{i=\tau+1}^n \phi_i \cdot \frac{\tau^2}{6}.$$

The first summation can be estimated up to  $\pm \epsilon D$  using our algorithm in  $O(1/\epsilon^2 + \log n)$  space. The second summation is  $\tau^2/6$  times using the number of distinct elements with frequency at least  $\tau + 1$  and thus can also be estimated up to  $\pm \epsilon D$  in  $O(1/\epsilon^2 + \log n)$  space.

- **Huber objective [21].** For a given parameter  $\tau$ , the Huber objective is

$$\sum_{i=1}^{\tau} \phi_i \cdot \frac{i^2}{2} + \sum_{i=\tau+1}^n \phi_i \cdot (\tau i - 1/2).$$

The first summation is can be estimated up to  $\pm \epsilon D$  using our algorithm in  $O(1/\epsilon^2 + \log n)$  space. The second summation can be written as  $\tau$  times the mass of elements with frequency at least  $\tau + 1$  minus half the number of distinct elements with frequency at least  $\tau + 1$ . So, in total, the objective can be estimated up to  $\pm \epsilon D$  in  $O(1/\epsilon^2 + \log n + \log \log m + \log(m/D))$  space.

To our knowledge, the best known previous algorithms for these problems use  $O(1/\epsilon^2 \log n)$  bits of space (by storing identities of sampled elements) or do not account for the space associated with randomness [15, 11, 12, 13]. In general, we can apply the guarantees of Equation (2) to estimate, in  $O(1/\epsilon^2 + \log n)$  space, any symmetric function which depends on a constant frequencies and is Lipschitz with respect to  $L_1$  error in the profile.

### 1.3 Technical Overview

For simplicity, we focus in this overview on the  $\pm \epsilon m$  guarantee of the algorithm in Equation (3). The same algorithm, but with slightly different parameters, achieves the guarantee in Equation (2). Our new analysis of the algorithm of Datar and Muthukrishnan is given in Appendix A. Although the algorithm is suboptimal, it illustrates the core issues that the techniques of optimal algorithm have to address. The method samples elements uniformly from the set of distinct elements in the stream and uses the (rescaled) empirical profile of the samples as the estimated profile  $\hat{\phi}$ . In the context of an  $L_1$  guarantee of  $\epsilon m$  additive error, we observe that it suffices to estimate only the  $\phi_i$ 's for  $i$  up to  $O(1/\epsilon)$ , as the remaining values can be set to zero without incurring much error (as there cannot be many high frequency elements). We then study the expected  $L_1$  estimation error of the first  $O(1/\epsilon)$  entries. The analysis crucially uses the specific properties of the profile function, leading to an  $O(1/\epsilon^2 \log(1/\epsilon))$  bound on the sample size. Naïvely, each of these sampled elements requires  $\log n + \log m$  bits to store its identity and count.

This algorithm can be improved using the fact that, to compute the profile, the actual identities of the sampled elements are not important as long as we can distinguish among them. This makes it possible to reduce the space by hashing sampled elements to a smaller universe of size that is quadratic in the sample size (quadratic dependence being necessary to avoid collisions). Since the sample size is polynomial in  $1/\epsilon$ , each hash can be represented using  $O(\log(1/\epsilon))$  bits. As we are only concerned with frequencies up to  $O(1/\epsilon)$ , the counts of each sampled element can also be stored in  $O(\log(1/\epsilon))$  bits, leading to an overall space bound of  $O(1/\epsilon^2 \log^2(1/\epsilon))$  bits. Although this algorithm is suboptimal, we believe that its simplicity makes it appealing in practice.

The optimal algorithm (Section 2) is much more technically involved. As with the previous algorithm, it hashes sampled elements into a smaller universe to reduce space. However, the size of the hash table is now linear, not quadratic, in the sample size. This removes the need to store the hashed identities as we can store the entire hash table explicitly. Combined with a more careful analysis of the number of bits required to represent the counts of sampled elements, this removes the “extra”  $\log(1/\epsilon)$  factor. This improvement, however, comes at the price of allowing collisions, meaning that elements with different frequencies are now mixed together, and the profile of hashed elements does not approximate the original one<sup>5</sup>. This necessitates inverting this mixing process to obtain frequency estimates for the original sample.

To simplify the analysis, we use the “Poissonization” trick so that outcomes in different buckets in the hash table are independent. Specifically, we use an additional hash function which maps an element to a  $\text{Poi}(1)$  random variable. We create that many distinct copies of each sampled element and add these copies to the hash table. Our goal is to use the hash table to estimate the number of sampled elements (before Poissonization) with frequency  $i$  for  $i \in \{1, \dots, O(1/\epsilon)\}$ . We achieve this via an iterative algorithm. Letting  $\hat{F}_j$  be our estimate for the number of distinct elements in our sample with frequency  $j$ , assume we are given the estimates  $\hat{F}_1, \dots, \hat{F}_{i-1}$  and want to estimate  $\hat{F}_i$ . We observe that there are two types of buckets in the hash table with count  $i$ . “Good” buckets are those which contain a single element with frequency  $i$ . “Bad” buckets are those which contain multiple elements (due to hash collisions) which sum to  $i$ .

To estimate the number of bad buckets, we sum, over all integer partitions of  $i$  with at least two summands, the estimated probability that that exact combination of elements hashes to the same bucket. These estimated probabilities come from our estimates  $\hat{F}_1, \dots, \hat{F}_{i-1}$  as well as our knowledge of the sample size compared to the size of the hash table. We can then estimate the number of good buckets by subtracting the estimated number of bad buckets from the total number of observed buckets with count  $i$ . Finally, we estimate  $\hat{F}_i$  by inverting the probability of getting a good bucket, i.e., of a bucket containing exactly one element.

This procedure produces the correct estimates under the assumption that the numbers of each type of bucket described above occur according to their expectations. As this is not the case, two types of error are introduced when calculating  $\hat{F}_i$ : random error due to deviations of the number of buckets with count  $i$  and propagation error due to using noisy estimates  $\hat{F}_1, \dots, \hat{F}_{i-1}$  in the calculation of the number of bad buckets with count  $i$ . At first glance, this second type of error has the potential to grow out of control as errors early on compound through the iterative estimation procedure. However, we show that the total propagation error across all coordinates of the estimated profile is within a constant factor of the total

---

<sup>5</sup> For example, a stream of all distinct elements will be hashed to one where a constant fraction of elements will have duplicates.



random error. To prove this result, we carefully analyze the sensitivity of the estimation of the number of bad buckets with count  $i$  to changes in the estimated numbers of elements with counts less than  $i$ . Early errors can compound as they recursively affect all subsequent estimates: error in estimating the number of elements with frequency  $i$  affects the estimate of the number of bad buckets for all counts greater than  $i$ . However, the propagation of these errors is limited by the fact that the probability of a large number of elements hashing to the same bucket decays exponentially. We ultimately bound the error introduced by allowing hash collisions to an additional  $O(\epsilon m)$  term in the expectation of  $\|\phi - \hat{\phi}\|_1$ .

One aspect of the algorithm that we have so far swept under the rug is how the algorithm samples elements: we need to adaptively maintain a sample of  $O(1/\epsilon^2 \log(1/\epsilon))$  elements. In a similar methodology to the optimal distinct elements sketch [26], we hash each element to a random identity in  $[n]$  and sample all elements which have least significant bit at least  $\ell$  after hashing. The variable  $\ell$  indicates the current sampling “level”. We track the stream length and number of distinct elements over time in order to update the level and maintain the correct number of samples. In order to remove the counts associated with elements that are no longer in the sample once the level updates, in each cell of the hash table, we keep separate counters stratified by the least significant bit of the hash of contributing elements.

The analysis of this algorithm requires pairwise independence of the counts of buckets. To ensure this pairwise independence holds after replacing truly random bits by a pseudorandom generator, we use Nisan’s pseudorandom generator. Since we need to preserve distributions of the counts of *pairs* of buckets, which are  $O(\log(1/\epsilon))$ -bit long, a random seed of length  $\text{polylog}(1/\epsilon)$  suffices. (Note that we cannot use Nisan’s generator to ensure that the bucket counts are fully independent, as that would require a random seed of length equal to the number of buckets, the space of our algorithm, times  $\log(1/\epsilon)$ ). We note that the technique of employing Nisan’s generator to achieve  $O(1)$ -wise independence introduced in this paper appears to be quite versatile, and has been since used for other streaming algorithms [25].

Our lower bound (Section 3) for algorithms achieving the guarantee of Equation (3) proceeds via a reduction from a direct sum of multiple instances, where each instance can be viewed as the composition of the Indexing problem with the Gap Hamming Distance problem with different parameters. To illustrate the basic connection between profile estimation and these communication problems, note that  $\phi_1$  can be used to count the number of elements which appear in exactly one of two binary strings to solve the Gap Hamming Distance problem<sup>6</sup> while distinguishing between there existing an element with frequency  $i$  or  $i - 1$  can be used to solve Indexing (by Bob adding  $i - 1$  copies of the element corresponding to his index).

The entries  $\phi_1, \phi_2 \dots \phi_{1/\epsilon}$  of the profile vector are split into “scales”, where each scale contains entries  $\phi_i$  for comparable (up to a constant factor) values of  $i$ . Intuitively, each scale contributes  $1/\epsilon^2$  term to the lower bound, for a total bound of  $\Omega(1/\epsilon^2 \log(1/\epsilon))$  bits. As there are known reductions of Gap Hamming Distance from the Indexing problem, we ultimately are able to prove our entire lower bound via an involved reduction from Indexing itself.

---

<sup>6</sup> In fact, this simple reduction is how we prove a lower bound for algorithms achieving the guarantee of Equation (2).

## 1.4 Discussion and Open Questions

We revisit the problem of estimating the profile of a data stream, a problem that appears commonly in practice and has applications to estimating symmetric functions of frequencies. We give space-optimal algorithms for two types of error guarantees. Our results focus on producing good estimates for entries of the profile corresponding to elements with small frequency (either explicitly through the parameter  $\tau$  in Theorem 1 or implicitly by letting the error scale with the mass of the stream in Theorem 2).

One direction for future work is to study profile estimation guarantees that put more emphasis on estimating large entries of the profile. What is the optimal space complexity of estimating the profile up to  $\pm\epsilon D$  on the first  $\tau$  coordinates for superconstant  $\tau$ ? Recall that if  $\tau > D^2$ ,  $\Omega(D \log m)$  bits of space are required. Estimation in terms of  $L_1$  error of the profile requires that we approximate the number of elements appearing exactly  $i$  times (for many  $i$ ). If we allow for approximating the number of elements appearing *approximately*  $i$  times, can we use less space? Answering these questions may imply improved algorithms for a broader class of symmetric functions.

The profile also appears in literature on distribution testing. Several works use the profile of a sample from a distribution to give sample-optimal testers for a broad class of symmetric properties (e.g., testing uniformity or estimating entropy) [31, 1, 9, 3]. For the right notion of error, a streaming algorithm for profile estimation may be able to be used to process a sample in sublinear space while retaining the performance of the testing algorithms. We leave the study of this as an intriguing open question.

### Paper Organization

The paper is organized as follows. In Section 2, we present and analyze our space-optimal algorithm. In Section 3, we present the lower bounds, showing the optimality of our algorithm. Finally, in Appendix A, we include the analysis of a simpler but suboptimal algorithm based on that of Datar and Muthukrishnan. Due to page limits, we defer many of the proofs to the full version.

## 2 Profile Estimation Algorithm

► **Theorem 1** ( $\pm\epsilon D$ ). *For any  $\epsilon > 0$  and  $\tau = O(1)$ , with input parameters  $B = \Theta(1/\epsilon^2)$  and errortype =  $D$ , Algorithm 1 uses  $O(1/\epsilon^2 + \log n)$  bits of space,  $O(\log(1/\epsilon) + \log \log n)$  expected amortized update time,  $O(1)$  post-processing time, and returns an estimated profile  $\hat{\phi}$  that satisfies*

$$\sum_{i=1}^{\tau} |\phi_i - \hat{\phi}_i| \leq \epsilon D$$

with probability 9/10.

► **Theorem 2** ( $\pm\epsilon m$ ). *For any  $\epsilon > 0$ , with input parameters  $B = \Theta(1/\epsilon^2 \log(1/\epsilon))$ ,  $\tau = O(1/\epsilon)$ , and errortype =  $m$ , Algorithm 1 uses  $O(1/\epsilon^2 \log(1/\epsilon) + \log n + \log \log m)$  bits of space,  $O(\log(1/\epsilon))$  expected amortized update time,  $O(1/\epsilon^3 \log(1/\epsilon))$  post-processing time, and returns an estimated profile  $\hat{\phi}$  that satisfies*

$$\sum_{i=1}^m |\phi_i - \hat{\phi}_i| \leq \epsilon m$$

with probability 9/10.



Before we describe and analyze Algorithm 1, we will give a few remarks on its inputs. In addition to the stream, the algorithm takes as input several parameters: the domain size  $n$  of the stream elements, a frequency threshold  $\tau$  (we will ignore counts that exceed this threshold), a number of buckets  $B$  for the core hash table, an error parameter  $\epsilon$ , and a variable *errortype* indicating whether the error guarantee will be  $\pm\epsilon D$  or  $\pm\epsilon m$ .

For the input parameter  $n$ , only an upper bound on the domain size is required to set the domain and range of the hash function  $g_1$  which is used to sample elements. As our bounds depend logarithmically in  $n$ , any poly( $n$ ) upper bound suffices and we will assume for simplicity it is a power of two. We will assume that  $B = \Omega(\log n)$ . If this is not the case, we can pick a smaller  $\epsilon$  so that  $B = \Theta(\log n)$ , paying an additive term of  $O(\log n)$  in space and leaving the asymptotic complexity unchanged. We also assume that  $B = O(D)$ : otherwise we have almost enough space to store the entire frequency histogram of the stream.

### Algorithm Description

The algorithm is decomposed into four parts: the main algorithm **EstimateProfile** (Algorithm 1), the sampling procedure **Sample** (Algorithm 2), the update procedure **IncrementCounters** (Algorithm 3), and the post-processing procedure **InvertCounts** (Algorithm 4). The core data structure maintained by the algorithm is an array  $A$  of  $B$  buckets. Each bucket will contain, as necessary, pairs of (level, counter) indicating the summed frequency of items which hash to that bucket with a certain sampling “level”.

Elements are sampled using the hash function  $g_1$  and the **Sample** subroutine. The main algorithm maintains a current level  $\ell_{cur}$  and a stream element  $x$  is sampled if the position of the least significant 1 bit in the binary representation of  $g_1(x)$  is at least  $\ell_{cur}$ . The level of the sampled element is the position of its least significant bit minus  $\ell_{cur}$ .  $\text{Poi}(1)$  copies of sampled elements are made and assigned to random locations in the hash table using hash functions  $g_2, z, h_1, \dots, h_H$ . The main algorithm periodically updates  $\ell_{cur}$  to reduce the sampling probability based on constant factor estimates of the number of distinct elements and stream length to ensure that the number of samples is correct.

For each copy of a sampled element, we update the bucket count in the corresponding bucket in the hash table via **IncrementCounters**. If a (level, counter) pair already exists in that bucket for the level of our sample, we increment the counter; otherwise, we create a new (level, counter) pair. If a counter ever exceeds  $\tau$ , we stop incrementing that counter. When the current level  $\ell_{cur}$  is incremented, we update all (level, counter) pairs in all buckets of the array by decrementing the level (remember the level of a sample is the position of the least significant bit relative to  $\ell_{cur}$ ). Any time the level goes below zero, we remove the corresponding pair from its bucket.

At the end of the stream, we observe the number of buckets with total counts (summed over all counters in the bucket)  $1, \dots, \tau$ , i.e., the profile of the array. In **InvertCounts**, we estimate the profile of the sampled elements (which have been corrupted by hash collisions) in an iterative process using dynamic programming. Using the number of nonempty bins, we estimate the probability that a bucket receives a single element. We call these “good” buckets as their counts correspond to a single sampled item of that frequency. Using this estimated probability, we can estimate the number of items with a given frequency from the number of observed good buckets with that count. Unfortunately, there also exist “bad” buckets with multiple items whose count is the sum of the frequency of those items. As there cannot be a bad bucket with count 1, we first estimate the number of sampled elements with frequency 1 and use that to estimate the number of bad buckets of frequency 2. These estimates are then used to estimate the number of bad buckets of frequency 3, and so on. The

---

**Algorithm 1** EstimateProfile.
 

---

**Input:** stream  $\mathbf{x} = x_1, \dots, x_m$ , domain size  $n$ , frequency threshold  $\tau$ , number of buckets  $B$ , error parameter  $\epsilon$ , *errortype* (either  $D$  or  $m$ )

**Output:** estimated profile  $\hat{\phi} = \hat{\phi}_1, \dots, \hat{\phi}_\tau$

- 1:  $T \leftarrow \Theta(B^2), H \leftarrow \Theta(\log B), K \leftarrow \Theta(1)$
- 2: Initialize a distinct elements sketch with relative error  $\epsilon/10$  [26] and, if *errortype* =  $D$ , also initialize a strong tracking distinct elements sketch with relative error  $1/10$  [5]
- 3: Initialize the main array  $A$  of  $B$  buckets as a variable-bit-length array [4] which will store in each bucket a string of (level, counter) pairs
- 4: Initialize hash functions  $g_1 : [n] \rightarrow [n], g_2 : [n] \rightarrow [T], h_1, \dots, h_H : [T] \rightarrow [B]$
- 5: Initialize hash function  $z : [T] \rightarrow \mathbb{N} \cup \{0\}$  that maps values in  $[T]$  to the outcome of a  $\text{Poi}(1)$  random variable
- 6: Initialize current level  $\ell_{cur} \leftarrow 1$
- 7: **for**  $x_t \in \mathbf{x}$  **do** ▷ Processing stream updates
- 8:   Update distinct elements sketches
- 9:   Let  $\tilde{D}_t$  be the estimate of the tracking sketch and let  $t$  be the stream length so far
- 10:   **if** (*errortype* =  $m$  and  $2^{\ell_{cur}} < \min\{tK/B, n\}$ ) or
- 11:    (*errortype* =  $D$  and  $2^{\ell_{cur}} < \tilde{D}_t K/B$ ) **then** ▷ Decrease sampling probability
- 12:      $\ell_{cur} \leftarrow \ell_{cur} + 1$
- 13:     For each (level, counter) pair in  $A$ , decrement the level
- 14:     If any level falls below 0, remove the corresponding pair from  $A$
- 15:   **end if**
- 16:    $\ell, a_1, \dots, a_H \leftarrow \text{Sample}(x_t, H, g_1, g_2, z, h_1, \dots, h_H)$
- 17:   **if**  $\ell \geq \ell_{cur}$  **then**
- 18:      $\text{IncrementCounters}(A, a_1, \dots, a_H, \ell - \ell_{cur}, \tau)$
- 19:   **end if**
- 20: **end for**
- 21:  $\hat{D} \leftarrow$  distinct elements estimate with error  $\epsilon/10$  ▷ Post-processing to estimate profile
- 22:  $G \leftarrow$  the number of nonempty buckets in  $A$
- 23:  $\hat{S} \leftarrow -B \ln(1 - \frac{G}{B})$  estimate of the number of elements which hash to 1 under  $g_1$
- 24:  $b_i \leftarrow$  number of buckets in  $A$  with total count  $i$  for  $i \in \{1, \dots, \tau\}$
- 25:  $\hat{F}_1, \dots, \hat{F}_\tau \leftarrow \text{InvertCounts}(B, \hat{S}, b_1, \dots, b_\tau)$  ▷ Estimate the profile of the sampled elements
- 26:  $\hat{\phi}_i \leftarrow \left(\frac{\hat{D}}{\hat{S}}\right) \hat{F}_i$  for  $i \in \{1, \dots, \tau\}$
- 27: **return**  $\hat{\phi}_1, \dots, \hat{\phi}_\tau$

---

dynamic program allows us to efficiently compute these iterative estimates without having to exhaustively list all integer partitions. Our final profile estimate in `EstimateProfile` comes from renormalizing the estimates of the empirical sample profile returned by `InvertCounts`.

### Notation

To introduce some notation, let  $D_t = |\{x \in (x_1, \dots, x_t)\}|$  be the number of distinct elements in the stream up to time  $t \in [m]$  with the shorthand  $D = D_m$ . Let  $\tilde{D}_t$  be the estimate of the tracking sketch at time  $t$ . Let  $\ell_t$  be the value of  $\ell_{cur}$ , the current level of the algorithm, at time  $t$ .

---

**Algorithm 2** Sample.

**Input:** stream element  $x$ , max copies  $H$ , hash functions  $g_1, g_2, z, h_1, \dots, h_H$   
**Output:** sampling level, buckets to update  $a_1, \dots, a_H$

- 1:  $\ell \leftarrow$  least significant nonzero bit of  $g_1(x)$
- 2:  $x' \leftarrow g_2(x)$  ▷ Smaller ID
- 3: **for**  $i \in [H]$  **do**
- 4:   **if**  $i \leq z(x')$  **then** ▷ Checking number of copies from Poissonization
- 5:      $a_i \leftarrow h_i(x')$
- 6:   **else**
- 7:      $a_i \leftarrow 0$
- 8:   **end if**
- 9: **end for**
- 10: **return**  $\ell, a_1, \dots, a_H$

---

**Algorithm 3** IncrementCounters.

**Input:** Array  $A$ , buckets  $a_1, \dots, a_H$  to increment, level  $\ell$ , max frequency  $\tau$

- 1:  $j \leftarrow$  smallest  $i \in [H]$  s.t.  $a_i = 0$  ▷ The number of buckets to update
- 2: **for**  $i = 1, \dots, j - 1$  **do**
- 3:   **if** there exists a (level, counter) pair with level  $\ell$  in  $A[a_i]$  **then**
- 4:     Increment the corresponding counter unless it exceeds  $\tau$
- 5:   **else**
- 6:     Add a new pair  $(\ell, 1)$  to  $A[a_i]$
- 7:   **end if**
- 8: **end for**

---

**Algorithm 4** InvertCounts.

**Input:** number of buckets  $B$ , estimated number of sampled elements  $\hat{S}$ , number of buckets  $b_i$  with count  $i$  for  $i \in \{1, \dots, \tau\}$   
**Output:** estimated counts  $\hat{F}_1, \dots, \hat{F}_\tau$

- 1: Initialize  $\tau \times \tau$  array DP
- 2:  $\text{DP}[1, 1] \leftarrow b_1 e^{\hat{S}/B}$
- 3: **for**  $i \in \{2, \dots, \tau\}$  **do**
- 4:    $\text{DP}[i, i] \leftarrow \max\{b_i e^{\hat{S}/B} - \sum_{x=1}^{\lfloor i/2 \rfloor} \text{DP}[i, x], 0\}$
- 5:   **for**  $x \in \{1, \dots, \lfloor i/2 \rfloor\}$  **do**
- 6:      $\text{DP}[i, x] \leftarrow \sum_{k=1}^{\lfloor i/x \rfloor - 1} \sum_{x'=x+1}^{i-kx} \text{DP}[i - kx, x'] \left( \frac{\text{DP}[x, x]}{B} \right)^k \frac{1}{k!}$
- 7:     **if**  $i = 0 \bmod x$  **then**
- 8:        $\text{DP}[i, x] \leftarrow \text{DP}[i, x] + \frac{\text{DP}[x, x]^{i/x}}{B^{i/x-1} (i/x)!}$
- 9:     **end if**
- 10:   **end for**
- 11: **end for**
- 12:  $\hat{F}_i \leftarrow \text{DP}[i, i]$  for  $i \in \{1, \dots, \tau\}$

---

## 32:12 Space-Optimal Profile Estimation with Applications to Symmetric Functions

Let  $S = \{x \in \mathbf{x} : g_1(x) = 1\}$  be the set of elements sampled by our algorithm at the end of the stream. Let  $m_S = \sum_{i=1}^m \mathbb{1}[g_1(x_i) = 1]$  be the mass of elements sampled by our algorithm. Let  $F_i = |\{x \in S : |\{j : x_j = x\}| = i\}|$  be the number of sampled elements with frequency  $i$ . Let  $G$  be the number of nonempty buckets in the array  $A$  at the end of the stream. We use  $\log(x)$  to denote the logarithm of  $x$  base 2.

In the rest of this section, we start by bounding several quantities used by our algorithm. Then, the bulk of the analysis in Section 2.1 focuses on the estimation error introduced by the post-processing procedure in Algorithm 4. To begin, we analyze our algorithm under the assumption that all of our hash functions are fully random and at the end in Section 2.2 show that our analysis only required limited randomness.

By the guarantees of [5] and [26], the estimate of  $\tilde{D}_t$  is correct up to relative error  $1/10$  at all points in the stream and the estimate of  $\hat{D}$  is correct up to relative error  $\epsilon/10$  at the end of the stream with small constant failure probability and using independent randomness from the rest of the algorithm. In what follows, we will condition on the success of these estimators.

► **Lemma 3.** *With constant probability, over all points in the stream  $t \in [m]$ , it holds that the space used to store  $A$  is  $O(B)$ .*

► **Lemma 4.** *With constant probability, the size of the final sample is bounded as follows:*

(a) *If errortype =  $m$ ,  $|S| = \Theta(\max\{DB/m, D/n\})$ .*

(b) *If errortype =  $D$ ,  $|S| = \Theta(B)$ .*

► **Lemma 5.** *With constant probability, the number of elements with frequency  $i$  in a given bucket is distributed as  $\text{Poi}(F_i/B)$ .*

Recall that  $\hat{S} = -B \ln(1 - \frac{G}{B})$  is our estimate of  $|S|$ .

► **Lemma 6.** *With constant probability,  $|\hat{S} - |S|| = O(\sqrt{|S|})$ .*

We will now prove that the  $L_1$  profile estimation error is small if we rescale the empirical profile of the sampled elements  $F_1, \dots, F_\tau$ . Recall that in the algorithm we only estimate this empirical profile (the bulk of the later analysis will focus on the quality of that estimation).

► **Lemma 7.** *For errortype =  $m$  if  $B = \Theta(\frac{\log \tau}{\epsilon^2})$ ,*

$$\sum_{i=1}^{\tau} \left| \phi_i - \frac{D}{|S|} F_i \right| = O(\epsilon m),$$

and for errortype =  $D$  and  $\tau = O(1)$  if  $B = \Theta(\frac{1}{\epsilon^2})$ ,

$$\sum_{i=1}^{\tau} \left| \phi_i - \frac{D}{|S|} F_i \right| = O(\epsilon D),$$

both holding with constant probability.

### 2.1 Inverting Counts

Let  $b_i$  be the number of buckets with count exactly  $i$ . Let  $r_i$  be the number of buckets with count  $i$  that are formed by collisions of elements with smaller frequencies. Let  $s_i$  be the number of buckets with count  $i$  that are formed by a single element with frequency  $i$  falling in that bucket and no other elements falling in that bucket. Note that  $b_i = r_i + s_i$ .

The core idea of our post-processing procedure is to recursively estimate  $F_i$  for  $i = 1, \dots, \tau$  by relating the expectation of  $F_i$  to the expectation of  $s_i$ . Using prior estimates  $\hat{F}_1, \dots, \hat{F}_{i-1}$ , we will approximate  $\mathbb{E}[r_i]$ . Then, using  $b_i$  as an estimate of  $\mathbb{E}[b_i]$ , we will plug in all of these estimates to solve for  $F_i$ . Along the way, errors will be introduced both due to random deviations as well as error which propagate from the fact that we do not know the true  $F_j$ 's for  $j < i$ . The core technical challenge of our analysis is to bound these errors.

Let  $X_{i,k} \sim \text{Poi}(F_i/B)$  be a random variable corresponding to the number of elements with count  $i$  in bucket  $k$ . By Poissonization,  $X_{i,k}$  is mutually independent of all  $X_{i',k'}$  for  $(i, k) \neq (i', k')$ . Let  $X_k = \sum_{i=1}^m X_{i,k} \sim \text{Poi}(|S|/B)$  be the random variable associated with the number of elements in bucket  $k$ .

$$\begin{aligned} \mathbb{E}[s_i] &= \sum_{k=1}^B \Pr(\text{bucket } k \text{ contains a unique element which has count } i) \\ &= \sum_{k=1}^B \Pr(X_{i,k} = 1) \prod_{j \in [m]: j \neq i} \Pr(X_{j,k} = 0) \\ &= \sum_{k=1}^B \frac{F_i}{B} \left( e^{-F_i/B} \right) \prod_{j \in [m]: j \neq i} e^{-F_j/B} \\ &= B \left( \frac{F_i}{B} \right) e^{-|S|/B} \\ &= F_i e^{-|S|/B} \end{aligned}$$

As  $s_i = b_i - r_i$ , we can express  $F_i$  as

$$F_i = \mathbb{E}[b_i e^{|S|/B}] - \mathbb{E}[r_i e^{|S|/B}]. \quad (4)$$

As the expectations of the  $b_i$ 's and  $r_i$ 's depend on the true values  $F_1, \dots, F_i$ , we cannot calculate them exactly. Rather, our estimate  $\hat{F}_i$  will be formed by plugging into Equation (4) the empirical count of  $b_i$  and the approximation  $\hat{r}_i$  of  $\mathbb{E}[r_i]$  formed using our previous estimates  $\hat{F}_1, \dots, \hat{F}_{i-1}$  in place of the true  $F_j$ 's. Further, as we do not know  $|S|$ , we will need to plug in an estimate  $\hat{S}$  wherever it appears.

We will now show that we can express  $\mathbb{E}[r_i]$  as a function of  $F_1, \dots, F_{i-1}$ . Let  $Y_i \sim \text{Poi}(F_i/B)$  be a random variable corresponding to the number of elements with count  $i$  in a given bucket. Let  $\mathbf{y} = y_1, \dots, y_m$  denote a vector corresponding to a specific assignment of how many distinct elements of counts  $1, \dots, m$  appear in a given bucket.

$$\begin{aligned} \mathbb{E}[r_i] &= B \Pr(\text{a bucket has summed count } i \text{ and at least two distinct elements}) \\ &= B \sum_{\mathbf{y}: (\sum_{j=1}^m y_j \geq 2) \wedge (\sum_{j=1}^m y_j \cdot j = i)} \prod_{j=1}^m \Pr(Y_j = y_j) \\ &= B \sum_{\mathbf{y}: (\sum_{j=1}^m y_j \geq 2) \wedge (\sum_{j=1}^m y_j \cdot j = i)} \prod_{j=1}^m \left( \frac{F_j}{B} \right)^{y_j} \frac{e^{-F_j/B}}{y_j!} \\ &= B e^{-|S|/B} \sum_{\mathbf{y}: (\sum_{j=1}^m y_j \geq 2) \wedge (\sum_{j=1}^m y_j \cdot j = i)} \prod_{j=1}^{i-1} \left( \frac{F_j}{B} \right)^{y_j} \frac{1}{y_j!} \end{aligned}$$

Let  $\hat{r}_i(\hat{F}_1, \dots, \hat{F}_{i-1})$  be the quantity we get by calculating  $\mathbb{E}[r_i e^{|S|/B}]$  under estimated parameters  $\hat{F}_1, \dots, \hat{F}_{i-1}$ :

$$\hat{r}_i(\hat{F}_1, \dots, \hat{F}_{i-1}) = B \sum_{\mathbf{y}: (\sum_{j=1}^{i-1} y_j \geq 2) \wedge (\sum_{j=1}^{i-1} y_j \cdot j = i)} \prod_{j=1}^{i-1} \left( \frac{\hat{F}_j}{B} \right)^{y_j} \frac{1}{y_j!} \quad (5)$$

Then, our final estimate for  $\hat{F}_i$  will be:

$$\hat{F}_i = \max\{b_i e^{\hat{S}/B} - \hat{r}_i(\hat{F}_1, \dots, \hat{F}_{i-1}), 0\}. \quad (6)$$

### 2.1.1 Dynamic Programming

Calculating these estimates naively requires  $\exp(\tau)$  time as  $\hat{r}_i(\cdot)$  in Equation (5) is a summation over integer partitions of  $i$ . Therefore, we use a dynamic program in Algorithm 4 to calculate this expression efficiently.

Given positive integral parameters  $j, x$ , the quantity of interest will be the expected number of buckets under  $\hat{F}_1, \dots, \hat{F}_{i-1}$  which have total (summed) count  $j$  and minimum frequency element with frequency  $x$ . We will use the following description of the dynamic program which is expressed equivalently in pseudocode in Algorithm 4:

$$\text{DP}[j, x] = \begin{cases} \hat{F}_j & \text{if } x = j \text{ and } j < i \\ \sum_{k=1}^{\lfloor j/x \rfloor - 1} \sum_{x'=x+1}^{j-kx} \text{DP}[j - kx, x'] \left( \frac{\hat{F}_x}{B} \right)^k \frac{1}{k!} & \text{if } x \leq \lfloor j/2 \rfloor \text{ and } j \neq 0 \pmod{x} \\ \sum_{k=1}^{\lfloor j/x \rfloor - 1} \sum_{x'=x+1}^{j-kx} \text{DP}[j - kx, x'] \left( \frac{\hat{F}_x}{B} \right)^k \frac{1}{k!} \\ + \left( \frac{\hat{F}_x^{j/x}}{B^{j/x-1}} \right) \frac{1}{(j/x)!} & \text{if } x \leq \lfloor j/2 \rfloor \text{ and } j = 0 \pmod{x} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

► **Lemma 8.**  $\hat{r}_i(\hat{F}_1, \dots, \hat{F}_{i-1}) = \sum_{x=1}^{\lfloor i/2 \rfloor} \text{DP}[i, x]$ .

### 2.1.2 Error Analysis

Let  $\zeta_i$  be a random variable for the error introduced by using our old estimates to evaluate  $\hat{r}_i$ :

$$\zeta_i = |\hat{r}_i(F_1, \dots, F_{i-1}) - \hat{r}_i(\hat{F}_1, \dots, \hat{F}_{i-1})|. \quad (8)$$

Let  $\gamma$  be a random variable for the rest of the error in estimating  $F_i$  due to randomness deviations in  $b_i$  as well as due to our error in approximating  $|S|$  as  $\hat{S}$ ,

$$\gamma_i = |\mathbb{E}[b_i] e^{|\hat{S}|/B} - b_i e^{\hat{S}/B}|. \quad (9)$$

Note that as  $F_i \geq 0$ , thresholding our estimate  $\hat{F}_i$  to always be at least zero only reduces the error:

$$|F_i - \hat{F}_i| \leq \gamma_i + \zeta_i.$$

► **Lemma 9.**  $\sum_{i=1}^{\tau} \zeta_i \leq \sum_{i=1}^{\tau} \gamma_i$ .

As the errors propagate (via composition of the  $f_j^i$ 's), their magnitudes diminish geometrically, allowing us to bound the total propagated error by the initial errors.

Now, we will bound the  $\gamma_i$  terms which, via Lemma 9, bound  $\sum_{i=1}^{\tau} |\hat{F}_i - F_i|$  up to constant factors.



► **Lemma 10.** *If  $\text{errortype} = m$  and  $B = \Theta(\log \tau / \epsilon^2)$ ,*

$$\sum_{i=1}^{\tau} \gamma_i = O\left(\frac{\log \tau}{\epsilon}\right),$$

*and if  $\text{errortype} = D$ ,  $\tau = O(1)$ , and  $B = \Theta(1/\epsilon^2)$ ,*

$$\sum_{i=1}^{\tau} \gamma_i = O\left(\frac{1}{\epsilon}\right),$$

*both holding with constant probability.*

This completes the analysis of inversion procedure, showing that Algorithm 4 returns a set of frequencies  $\hat{F}_1, \dots, \hat{F}_\tau$  close in  $L_1$  distance to the empirical profile of our samples  $F_1, \dots, F_\tau$ .

## 2.2 Pseudorandomness

So far, in our analysis, we have assumed that our hash functions are fully random. In order to bound the space required to store the hash functions, we will now argue that the guarantees of the algorithm only require limited independence. We will require a lemma from prior work on applying Nisan's PRG to streaming algorithms.

► **Lemma 11** (Lemma 3 from [22]). *Consider an algorithm  $\mathcal{A}$  that, given a stream  $\mathcal{S}$  of elements  $x$ , and a function  $f : [n] \times \{0, 1\}^R \rightarrow [\text{poly}(M)] \times [\text{poly}(M)]$ , does the following:*

- *Set  $\mathcal{O} = (0, 0)$ ; Initialize length- $R$  chunks  $R_0, \dots, R_n$  of independent random bits*
- *For each new element  $x$ , perform  $\mathcal{O} = \mathcal{O} + f(x, R_x)$*
- *Output  $A(\mathcal{S}) = \mathcal{O}$*

*Assume that the function  $f(\cdot, \cdot)$  is computed by an algorithm using  $O(C + R)$  space and  $O(T)$  time. Then there is an algorithm  $\mathcal{A}'$  producing output  $\mathcal{A}'(\mathcal{S})$ , that uses only  $O(C + R + \log(Mn))$  bits of storage and  $O([C + R + \log(Mn)] \log(nR))$  random bits, such that*

$$\Pr[\mathcal{A}(\mathcal{S}) \neq \mathcal{A}'(\mathcal{S})] \leq 1/\text{poly}(n)$$

*over some joint probability space of randomness of  $\mathcal{A}$  and  $\mathcal{A}'$ . Then, the algorithm  $\mathcal{A}'$  uses  $O(T + \log(nR))$  arithmetic operations per each element  $x$ .*

Note several minor changes from the lemma as it appears in [22]: the original lemma includes a parameter for the size of stream updates which we omit as all of our updates are increments, the original lemma considers a one dimensional output while we consider a two dimensional output, and the original lemma bounds the bias by  $1/n$  while the bias can in fact be bounded by  $1/\text{poly}(n)$  without changing the asymptotic result.

► **Lemma 12.**  *$O(\log^3(B) + \log n)$  bits of randomness suffice for the hash functions used in Algorithm 1.*

**Proof.** The randomness of Algorithm 1 is realized through the following hash functions  $g_1, g_2, z, h_1, \dots, h_H$ . The hash function  $g_1 : [n] \rightarrow [n]$  is used to sample elements based on the least significant bit of their hashed values. The randomness of this hash function is used in analysis of the space needed to store  $A$  as well as  $|S|$  and  $F_1, \dots, F_m$ . The analysis of those quantities only use first and second moments, so pairwise independence suffices for  $g_1$ .

## 32:16 Space-Optimal Profile Estimation with Applications to Symmetric Functions

The hash function  $g_2 : [n] \rightarrow [T]$  maps the sampled elements down to a domain of size  $\Theta(B^2)$ . We only require that with constant probability, there are no collisions between sampled elements. As with constant probability  $|S| = O(\sqrt{T})$  (see Lemma 4), this holds under pairwise independence.

The hash function  $z : [T] \rightarrow \mathbb{N} \cup \{0\}$  maps sampled stream elements to the outcome of a  $\text{Poi}(1)$  random variable.<sup>7</sup> As long as  $z$  is pairwise independent, the total number and mass of (copied) samples will be concentrated about their expectation.

The hash functions  $h_1, \dots, h_H : [T] \rightarrow [B]$  map sampled stream elements to a bucket in the array  $A$ . These hash functions (as well as  $z$ ) affect  $b_i$ , the number of buckets with count exactly  $i$  (as well as their sum,  $G$ , the number of nonempty buckets).

In our analysis, we use the first and second moments of the  $b_i$ 's to bound the error of our algorithm. Let  $X_k$  be a random variable for the count in bucket  $k$ . The first two moments of  $b_i$  depend only on the joint distribution of  $(X_k, X_{k'})$ . We will use Lemma 11 to show that limited randomness suffices to simulate this distribution up to small bias. Consider the following setting of the parameters in Lemma 11 where the algorithm  $\mathcal{A}$  computes the pair  $(X_k, X_{k'})$  given fully random hash functions:

- $n = O(B^2 \log^2 \tau)$  as we are conditioning on the outcomes of  $g_1, g_2$  which have generated  $S$  distinct items for our stream coming from a domain of size  $O(B^2 \log^2 \tau)$ .
- $R = O(\log^2 B)$  is the number of random bits required to generate fully random  $z(x)$ ,  $h_1(x), \dots, h_H(x)$  as the range of these hash functions is  $B$  and there are  $H = O(\log B)$  such hash functions.
- $M = \tau$  as the counts of a bucket  $X_k$  is at most  $\tau$ .
- $f(x, R_x)$  is computed by  $\mathcal{A}$  as follows. Set the algorithm's output to be  $\mathcal{O} = (a, b) = (0, 0)$ .  $\mathcal{A}$  uses the initial bits of  $R_i$  to generate  $z(x)$ . Then,  $\mathcal{A}$  uses each of the next chunks of  $O(\log(B))$  bits in  $R_i$  to generate  $h_1(x), \dots, h_H(x)$ . On generating  $h_i(x)$ ,  $\mathcal{A}$  first checks if  $z(x) < i$  and if so, stops early and returns  $(a, b)$ . Otherwise,  $\mathcal{A}$  checks if  $h_i(x) = k$  and if so increments  $a$  by one (unless the count exceeds  $\tau$ ). Finally,  $\mathcal{A}$  checks if  $h_i(x) = k'$  and if so increments  $b$  by one (unless the count exceeds  $\tau$ ). Via this process,  $\mathcal{A}$  will recover the true counts of  $(X_k, X_{k'})$ .
- $C$ , the space used by  $\mathcal{A}$ , is  $O(\log B)$ . In processing a given stream element,  $\mathcal{A}$  stores the outcome of  $z(x)$ , the current  $h_i(x)$ ,  $\mathcal{O}$ , and the current increment to  $\mathcal{O}$ . In total, these are a constant number of quantities, each taking up at most  $O(\log B)$  space.

It follows from application of Lemma 11 that the algorithm  $\mathcal{A}'$  using Nisan's PRG requires only

$$O(C + R + \log(Mn)) \log(nR) = O(\log B + \log^2(B) + \log B) \log B = O(\log^3 B)$$

random bits to approximate the joint distribution of  $(X_k, X_{k'})$  up to bias  $\text{poly}(\epsilon)$ :

$$\Pr[\mathcal{A}(\mathcal{S}) \neq \mathcal{A}'(\mathcal{S})] \leq 1/\text{poly}(B).$$

For large enough  $\text{poly}(B)$ , this implies that the expectations and variances of the  $b_i$ 's using limited randomness will be correct up to a small factor of  $\epsilon$ , which is enough for our analysis.

As the hash functions  $g_1$  and  $g_2$  only require constant independence, they can be stored and queried in  $O(\log n)$  bits [8]. So, the total space required for storing randomness is  $O(\log^3(B) + \log n)$ . ◀

<sup>7</sup> For example via inverse transform sampling [https://en.wikipedia.org/wiki/Inverse\\_transform\\_sampling](https://en.wikipedia.org/wiki/Inverse_transform_sampling).

## 2.3 Putting it all together

We are now ready to prove the main theorems. In the interest of space, we present the proof for only for the  $\pm\epsilon m$  guarantee. The proof of the  $\pm\epsilon D$  guarantee follows a very similar argument and appears in the full version.

### Proof of Theorem 2.

**Correctness.** Recall that our estimator is

$$\hat{\phi}_i = \frac{\hat{D}}{\hat{S}} \hat{F}_i.$$

From Lemmas 9 and 10, with constant probability,

$$\sum_{i=1}^{\tau} |F_i - \hat{F}_i| = O\left(\frac{\log \tau}{\epsilon}\right).$$

Also, recall that by Lemma 7, with constant probability,

$$\sum_{i=1}^{\tau} \left| \phi_i - \frac{D}{|S|} F_i \right| = O(\epsilon m). \quad (10)$$

With constant probability,  $|S| = \Omega\left(\frac{D \log \tau}{m \epsilon^2}\right)$  by Lemma 4,  $|\hat{S} - |S|| = O(\sqrt{|S|})$  by Lemma 6, and  $|\hat{D} - D| \leq \epsilon D/10$  by the correctness of the distinct elements sketch (e.g., [26]). In addition,  $\log \tau / \epsilon = O(\epsilon B) = o(\epsilon m)$ . Under these conditions,

$$\begin{aligned} \sum_{i=1}^{\tau} \left| \frac{D}{|S|} F_i - \frac{\hat{D}}{\hat{S}} \hat{F}_i \right| &\leq \sum_{i=1}^{\tau} \frac{D}{|S|} |F_i - \hat{F}_i| + \left| \frac{D}{|S|} - \frac{\hat{D}}{\hat{S}} \right| \hat{F}_i \\ &\leq \frac{D}{|S|} \cdot O\left(\frac{\log \tau}{\epsilon}\right) + \sum_{i=1}^{\tau} \frac{\epsilon D}{5 \hat{S}} \hat{F}_i \\ &= O\left(\frac{m \epsilon^2}{\log \tau} \left(\frac{\log \tau}{\epsilon}\right)\right) + \frac{\epsilon D}{5} \\ &= O(\epsilon m). \end{aligned} \quad (11)$$

Recall that  $\tau = O(1/\epsilon)$ . Implicitly, we will predict  $\hat{\phi}_i = 0$  for all  $i > \tau$ . As there are at most  $m/\tau$  elements with frequency greater than or equal to  $\tau$ , this contributes error at most  $O(\epsilon m)$ . So, with appropriately chosen constants and with constant probability, triangle inequality with Equation (10) and Equation (11) gives

$$\sum_{i=1}^m |\phi_i - \hat{\phi}_i| \leq \epsilon m,$$

as required.

**Space.** Now, we will analyze the space used by the algorithm while processing stream updates. By Lemma 3, the array  $A$  can be maintained in  $O(B) = O(\log(1/\epsilon)/\epsilon^2)$  bits of space. By Lemma 12, it suffices to store  $O(\log^3(B) + \log n) = O(\log^3(1/\epsilon) + \log n)$  bits of randomness. As each stream update occurs, we must store its identity in  $O(\log n)$  bits of space. Storing the length of the stream naively takes  $O(\log m)$  bits. Given a  $\text{poly}(m)$  upper bound on the stream length, this can be reduced to  $O(\log \log m)$  bits using the Morris+

algorithm of [29] to maintain a small constant approximation to the stream length (which is all we require) at all times with small constant failure probability. The distinct elements sketch can be maintained in  $O(1/\epsilon^2 + \log n)$  bits of space [26]. Therefore, the total space usage of our algorithm is, with constant probability,  $O(\log(1/\epsilon)/\epsilon^2 + \log n + \log \log m)$  bits.

**Update time.** Each time a stream element appears, hashing it for sampling and to compress its ID each require a constant number of operations. Without accounting for the pseudorandom generator, Poissonization and incrementing bucket counts takes  $O(1)$  expected amortized time as each element is copied  $O(1)$  times in expectation and array updates can be done in expected amortized constant time [4] (by also storing the (level, counter) dictionaries in cells of the array  $A$  as variable-bit-length dictionaries). Deletions of (level, counter) pairs take constant amortized time: they take  $O(B)$  time each time the size of the stream doubles relative to  $B$ . Then, by Lemma 11 and the parameters used in Lemma 12, these updates take  $O(1 + \log(B^2 \log^2 B \log^2 \tau)) = O(\log(1/\epsilon))$  time accounting for Nisan's PRG. Finally, updating the distinct elements sketch takes  $O(1)$  time [26]. So, the total update time is  $O(\log(1/\epsilon))$ .

**Post-processing.** The dynamic program for post-processing maintains  $O(1/\epsilon^2)$  cells, and the computation takes time polynomial in  $1/\epsilon$ . Furthermore, all entries in the DP table are positive. It follows that performing the computation with  $O(\log(1/\epsilon))$  bits of precision per cell suffices to calculate the answer with a multiplicative error of  $1 + \epsilon^{O(1)}$ . So, the total space usage is  $O(\log(1/\epsilon)/\epsilon^2)$ .

For a given cell in the dynamic program in Algorithm 4, calculating  $\text{DP}[i, x]$  requires  $O(i^2/x)$  operations due to the nested sum. As we are only concerned with cells where  $x \leq i$ , filling the row in the table corresponding to a fixed  $i$  takes  $\sum_x O(i^2/x) = O(i^2 \log i)$  time. Summing over all  $i \in \{1, \dots, \tau\}$ , the total post-processing time is bounded by  $O(\log(1/\epsilon)/\epsilon^3)$ . ◀

### 3 Lower Bounds

For the two estimation guarantees we consider, we give lower bounds, showing that the space used by our algorithms is necessary. The proofs of these theorems utilize reductions from the IND problem in communication complexity and are deferred to the full version.

► **Theorem 13.** *Any single pass streaming algorithm which outputs an additive  $\epsilon D$  approximation to  $\phi_1$  with success probability at least  $9/10$  requires  $\Omega(1/\epsilon^2)$  bits of memory.*

► **Theorem 14.** *Any single pass streaming algorithm which outputs an additive  $\epsilon m L_1$  approximation to the profile with success probability at least  $99/100$  requires  $\Omega(\log(1/\epsilon)/\epsilon^2)$  bits of memory.*

---

#### References

- 1 Jayadev Acharya, Hirakendu Das, Alon Orlitsky, and Ananda Theertha Suresh. A unified maximum likelihood approach for optimal distribution property estimation. In *International Conference on Machine Learning*. PMLR, 2017.
- 2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237823.

- 3 Nima Anari, Moses Charikar, Kirankumar Shiragur, and Aaron Sidford. The bethe and sinkhorn permanents of low rank matrices and implications for profile maximum likelihood. In *Conference on Learning Theory*, pages 93–158. PMLR, 2021.
- 4 Daniel K. Blandford and Guy E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Trans. Algorithms*, 4:17:1–17:25, 2008.
- 5 Jaroslaw Blasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020. doi:10.1145/3309193.
- 6 Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch 12-heavy-hitters on sliding windows. *Theoretical Computer Science*, 554:82–94, 2014.
- 7 Luciana S. Buriol, Debora Donato, Stefano Leonardi, and Tobias Matzner. Using data stream algorithms for computing properties of large graphs. In *Workshop on Massive Geometric Data Sets (MASSIVE'05)*, pages 9–14, 2005.
- 8 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.
- 9 Moses Charikar, Kirankumar Shiragur, and Aaron Sidford. Efficient profile maximum likelihood for universal symmetric property estimation. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 780–791, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316398.
- 10 Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.
- 11 Edith Cohen. Stream sampling for frequency cap statistics. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 159–168, 2015.
- 12 Edith Cohen. Hyperloglog hyperextended: Sketches for concave sublinear frequency statistics. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 105–114, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3097983.3098020.
- 13 Edith Cohen and Ofir Geri. Sampling sketches for concave sublinear functions of frequencies. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1361–1371, 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/61b4a64be663682e8cb037d9719ad8cd-Abstract.html>.
- 14 Graham Cormode, Senthilmurugan Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, volume 5, pages 25–36, 2005.
- 15 Mayur Datar and S Muthukrishnan. Estimating rarity and similarity over data stream windows. In *European Symposium on Algorithms*, pages 323–335. Springer, 2002.
- 16 Charlie Dickens. Personal communication, 2023.
- 17 Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ics*, pages 66–80, 2010.
- 18 Guy Feigenblat, Ely Porat, and Ariel Shiftan. Exponential time improvement for min-wise based algorithms. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 57–66. SIAM, 2011.
- 19 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.
- 20 Badih Ghazi, Ben Kreuter, Ravi Kumar, Pasin Manurangsi, Jiayu Peng, Evgeny Skvortsov, Yao Wang, and Craig Wright. Multiparty reach and frequency histogram: Private, secure, and practical. *Proceedings on Privacy Enhancing Technologies*, 2022:373–395, January 2022. doi:10.2478/popets-2022-0019.
- 21 Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. doi:10.1214/aoms/1177703732.

- 22 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.
- 23 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208, 2005.
- 24 Rajesh Jayaram, David P. Woodruff, and Samson Zhou. Truly perfect samplers for data streams and sliding windows. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '22*, pages 29–40, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3524139.
- 25 Praneeth Kacham, Rasmus Pagh, Mikkel Thorup, and David P. Woodruff. Pseudorandom hashing for space-bounded computation with applications to streaming. In *Proceedings of the 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2023.
- 26 Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52, 2010.
- 27 Vijay Karamcheti, Davi Geiger, Zvi Kedem, and S Muthukrishnan. Detecting malicious network traffic using inverse distributions of packet contents. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 165–170, 2005.
- 28 Robert H. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.
- 29 Jelani Nelson and Huacheng Yu. Optimal bounds for approximate counting. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '22*, pages 119–127, New York, NY, USA, 2022. Association for Computing Machinery.
- 30 William J. J. Rey. *Introduction to Robust and Quasi-Robust Statistical Methods*. Universitext. Springer, Berlin, Heidelberg, 1983.
- 31 Gregory Valiant and Paul Valiant. Estimating the unseen: improved estimators for entropy and other properties. *Journal of the ACM (JACM)*, 64(6):1–41, 2017.

## **A** Re-analyzing the Algorithm of Datar and Muthukrishnan

In this section, we re-analyze the Datar-Muthukrishnan algorithm in the case where the error function is measured using the  $L_1$  norm according to Equation (3) where our goal is to estimate the profile up to error  $\pm \epsilon m$ .

Recall that the algorithm selects the set  $S$  of  $s$  samples uniformly at random from the support of the stream and computes exact frequencies of each element sampled. Let  $D$  be the number of distinct elements in the stream and let  $F_i$  be the number of samples with frequency  $i$ . The algorithm then estimates the ratio  $\phi_i/D$  by  $\frac{1}{s}F_i$ .

In our case, we make the following small modifications. First, since our goal is to estimate  $\phi_i$  as opposed to  $\phi_i/D$ , we run a distinct elements streaming algorithm [26] in parallel to get an estimate  $\hat{D}$  of the number of distinct elements up to a  $(1 \pm \epsilon)$  multiplicative factor. This requires  $O(1/\epsilon^2 + \log n)$  space, which is subsumed by the overall space bound. Then, our estimate of  $\phi_i$  will be  $\hat{\phi}_i = \frac{\hat{D}}{s}F_i$ . Since replacing  $D$  by  $\hat{D}$  changes the estimates by only a  $(1 \pm \epsilon)$  multiplicative factor, in the rest of this section we assume the algorithm knows  $D$  exactly.

Second, for all  $i > 2/\epsilon$ , we set  $\hat{\phi}_i$  to zero instead of  $\frac{D}{s}F_i$ . (As we note below, this adds only  $O(\epsilon m)$  to the total error bound). We refer to this procedure as the *modified Datar-Muthukrishnan algorithm*.

► **Theorem 15.** *The modified Datar-Muthukrishnan algorithm, with sample size  $s = O(1/\epsilon^2 \log(1/\epsilon))$ , returns an estimated profile vector  $\hat{\phi}$  such that  $\|\phi - \hat{\phi}\|_1 \leq \epsilon m$  with constant probability (say, at least  $2/3$ ).*



**Proof.** First, we note that estimating all  $\phi_i$  for  $i > 2/\epsilon$  as zero contributes at most  $\epsilon m/2$  to the  $L_1$  error of the estimate. This is because the total number of elements with frequency at least  $2/\epsilon$  is at most  $\epsilon m/2$ .

Lemma 7 bounds the  $L_1$  error of the reweighted empirical profile. In particular, if  $s = \Omega\left(\frac{D \log(1/\epsilon)}{m \epsilon^2}\right)$ ,

$$\sum_{i=1}^{\lceil 2/\epsilon \rceil} \left| \phi_i - \frac{D}{s} F_i \right| = O(\epsilon m).$$

As  $D \leq m$ ,  $s = \Theta(1/\epsilon^2 \log(1/\epsilon))$  suffices to achieve expected error of, say,  $\epsilon m/6$ . Markov's inequality completes the proof.  $\blacktriangleleft$

We note that the above algorithm, for each sample, stores  $\log n + \log m$  bits to maintain the identity of the sample as well as its count for a total space complexity  $O(1/\epsilon^2 \log(1/\epsilon) \log(nm))$  bits. In the following section, we present an algorithm which uses  $O(1/\epsilon^2 \log^2(1/\epsilon) + \log n)$  bits, avoiding this multiplicative  $\log(nm)$  dependence.

## Improving Storage through Hashing

In this section we outline an improved algorithm with a reduced space bound of  $O(1/\epsilon^2 \log^2(1/\epsilon) + \log n)$ , i.e., replacing  $\log(nm)$  with  $\log(1/\epsilon)$ . Although this bound is still not optimal, the algorithm in this section will help us illustrate the challenges in obtaining the optimal bound.

First, recall from the analysis in the previous section that a more fine-tuned bound on the number of required samples is  $s = O(1/\epsilon^2 \log(1/\epsilon) \cdot D/m)$ , so as the number of distinct elements decreases, we need fewer samples. Let  $C, C' > 1$  be sufficiently large constants. Consider the following algorithm for processing a stream element  $x_i$  given  $\epsilon, m$  and two hash functions  $g$  and  $h$ :

1. **Sampling:** Hash  $x_i$  using  $g$  to the universe  $[\frac{m}{C'/\epsilon^2 \log(1/\epsilon)}]$  uniformly at random.
  - If  $g(x_i) = 1$ , continue to step 2.
  - Else, ignore  $x_i$ .
2. **Compression:** Hash  $x_i$  using  $h$  to the universe  $[C(1/\epsilon^2 \log(1/\epsilon))^2]$ 
  - Insert/increment the count of  $h(x_i)$  in a dictionary (sparse hash table)  $H$
  - If the count of  $h(x_i)$  exceeds  $2/\epsilon$ , marks its count as N/A and ignore it going forward

Call  $S$  the set of unique stream elements that hash to 1 under  $g$ . After all items are inserted, we use the dictionary  $H$  to compute an empirical estimate  $\hat{\phi}$  of the profile (rescaled by  $D/|S|$ ), as in the previous algorithm. In what follows, we show that the resulting estimated profile  $\hat{\phi}$  is within an  $L_1$  distance of  $\epsilon m$  from the true profile vector  $\phi$ , with constant probability.

First, we will show that with high constant probability,  $|S|$  is at least  $1/\epsilon^2 \log(1/\epsilon) D/m$ . The probability that any element lands in  $S$  is  $\frac{C'/\epsilon^2 \log(1/\epsilon)}{m}$  and there are  $D$  elements, so the expected number of samples is as desired. As the distribution of  $|S|$  is binomial, the variance is at most the expectation and therefore the size of the sample is correct up to constant factors with constant probability via Chebyshev's inequality.

Now, consider the hash table  $H$ . If there are no collisions in the hash table, its nonzero entries (those stored in the dictionary) will contain the true counts of each element in  $S$  (ignoring elements with counts exceeding  $2/\epsilon$ , which is fine since we are not counting those). Note that  $|S| < 10C'/\epsilon^2 \log(1/\epsilon)$  with a large constant probability as  $D \leq n$ , so  $|H| = C/(10C')^2 |S|^2$ . For large enough constant  $C$ , it is unlikely for there to be any collisions, and  $H$  contains the appropriate number of samples along with their true counts.

**Space**

It takes  $\log n$  bits to store  $x_i$ . The rest of the space is taken up by the hash table. There are  $|S|$  elements in the hash table, which we have already argued is at most  $O(1/\epsilon^2 \log(1/\epsilon))$ . For each element, we must store its identity  $h(x_i)$  as well as the corresponding count, both of which take up  $O(\log(1/\epsilon))$  bits. In our analysis, we only ever require pairwise independence, so the hash functions can be stored in  $O(\log n)$  space. Therefore, the total space in bits of this construction is

$$O(1/\epsilon^2 \log^2(1/\epsilon) + \log n)$$

**Improving the bound**

It can be seen that the “extra”  $\log(1/\epsilon)$  factor in the  $O(1/\epsilon^2 \log^2(1/\epsilon))$  bound is mostly due to the need for avoiding collisions of the sampled elements  $S$  under the hash function  $h$  (i.e., ensuring that  $h$  is perfect)<sup>8</sup>. This requires storing  $O(\log(1/\epsilon))$  bits per sample, to disambiguate distinct elements in  $S$ . The algorithm presented in Section 2 achieves the optimal space by hashing elements in  $S$  to a hash table of size  $O(|S|)$ , not  $O(|S|^2)$ , removing the need to store hashed IDs. This, however, comes at the price of allowing collisions, which means that elements with different frequencies are mixed together. Iteratively “inverting” this mixing process to obtain frequency estimates is the most technically challenging part of our algorithm.

---

<sup>8</sup> In addition, we also need to maintain the count of each sampled element, which also takes  $O(\log(1/\epsilon))$  bits per sample. However, this issue can be solved more easily.