2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Perfect L_p Sampling in a Data Stream

Rajesh Jayaram
Computer Science Department
Carnegie Mellon University
Pittsburgh, U.S.A.
rkjayara@cs.cmu.edu

David P. Woodruff
Computer Science Department
Carnegie Mellon University
Pittsburgh, U.S.A.
dwoodruf@cs.cmu.edu

***Abstract*—In this paper, we resolve the one-pass space complexity of L_p sampling for $p \in (0,2)$. Given a stream of updates (insertions and deletions) to the coordinates of an underlying vector $f \in \mathbb{R}^n$, a *perfect* L_p sampler must output an index i with probability $|f_i|^p/\|f\|_p^p$, and is allowed to fail with some probability δ. So far, for $p > 0$ no algorithm has been shown to solve the problem exactly using poly$(\log n)$-bits of space. In 2010, Monemizadeh and Woodruff introduced an *approximate* L_p *sampler*, which outputs i with probability $(1 \pm \nu)|f_i|^p/\|f\|_p^p$, using space polynomial in ν^{-1} and $\log(n)$. The space complexity was later reduced by Jowhari, Sağlam, and Tardos to roughly $O(\nu^{-p}\log^2 n \log \delta^{-1})$ for $p \in (0,2)$, which tightly matches the $\Omega(\log^2 n \log \delta^{-1})$ lower bound in terms of n and δ, but is loose in terms of ν.**

Given these nearly tight bounds, it is perhaps surprising that no lower bound at all exists in terms of ν—not even a bound of $\Omega(\nu^{-1})$ is known. In this paper, we explain this phenomenon by demonstrating the existence of an $O(\log^2 n \log \delta^{-1})$-bit *perfect* L_p sampler for $p \in (0,2)$. This shows that ν need not factor into the space of an L_p sampler, which completely closes the complexity of the problem for this range of p. For $p = 2$, our bound is $O(\log^3 n \log \delta^{-1})$-bits, which matches the prior best known upper bound of $O(\nu^{-2}\log^3 n \log \delta^{-1})$, but has no dependence on ν. Finally, we show that a $(1 \pm \epsilon)$ relative error estimate of the frequency f_i of the sampled index i can be obtained using an additional $O(\epsilon^{-p}\log n)$-bits of space for $p < 2$, and $O(\epsilon^{-2}\log^2 n)$ bits for $p = 2$, which was possible before only by running the prior algorithms with $\nu = \epsilon$.

***Index Terms*—streaming; sampling; hashing; sketching**

For a full version of this paper see https://arxiv.org/abs/1808.05497

I. Introduction

The streaming model of computation has become increasingly important for the analysis of massive datasets, where the sheer size of the input imposes stringent restrictions on the resources available to algorithms. Examples of such datasets include internet traffic logs, sensor networks, financial transaction data, database logs, and scientific data streams (such as huge experiments in particle physics, genomics, and astronomy). Given their prevalence, there is a large body of literature devoted to designing extremely efficient one-pass algorithms for analyzing data streams. We refer the reader to [2], [46] for surveys of these algorithms and their applications.

More recently, the technique of sampling has proven to be tremendously powerful for the analysis of data streams. Substantial literature has been devoted to the study of sampling for problems in big data [46], [28], [8], [11], [12], [10], [18], [23], [38], [43], [53], [9], [22], [21], with applications to network traffic analysis [50], [31], [26], [42], [17], databases [49], [28], [29], [30], [41], [40], distributed computation [55], [14], [15], [51], and low-rank approximation [55], [20], [16]. While several models for sampling in data streams have been proposed [3], [1], [14], perhaps the most widely studied are the L_p samplers introduced in [45]. Roughly speaking, given a vector $f \in \mathbb{R}^n$, the goal of an L_p sampler is to return an index $i \in \{1,2,\ldots,n\}$ with probability $|f_i|^p/\|f\|_p^p$. In the data stream setting, the vector f is given by a sequence of updates (insertions or deletions) to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where Δ can either be positive or negative. A 1-pass L_p sampler must return an index given only one linear pass through the updates of the stream.

Since their introduction, L_p samplers have been utilized to develop alternative algorithms for many important streaming problems, such as the heavy hitters problem, L_p estimation, cascaded norm estimation, and finding duplicates in data streams [1], [45], [35], [5]. For the case of $p = 1$ and insertion only streams, where the updates to f are strictly positive, the problem is easily solved using $O(\log n)$ bits of space with the well-known reservoir sampling algorithm [52]. When deletions to the stream are allowed or when $p \neq 1$, however, the problem is more complicated. In fact, the question of whether such samplers even exist was posed by Cormode, Murthukrishnana, and Rozenbaum in [13]. Later on, Monemizadeh and Woodruff demonstrated that, if one permits the sampler to be *approximately* correct, such samplers are indeed possible [45]. We formally state the guarantee given by an approximate L_p sampler below.

Definition 1. Let $f \in \mathbb{R}^n$ and $\nu \in [0,1)$. For $p > 0$, an *approximate L_p sampler* with ν-relative error is an algorithm which returns an index $i \in \{1,2,\ldots,n\}$ such that for every $j \in \{1,2,\ldots,n\}$

$$\Pr[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm \nu) + O(n^{-c})$$

Where $c \geq 1$ is some arbitrarily large constant. For $p = 0$, the problem is to return j with probability $(1 \pm \nu)\max\{1, |f_j|\}/|\{j \ : \ f_j \neq 0\}| + O(n^{-c})$. If $\nu = 0$, then the sampler is said to be *perfect*. An L_p sampler is allowed to output `FAIL` with some probability δ, however in this case it must not output any index.

The authors thank the partial support by the National Science Foundation under Grant No. CCF-1815840.

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00058

L_p sampling upper bound (bits)	p range	Notes	Citation
$O(\log^3(n))$	$p = 0$	perfect L_0 sampler, $\delta = 1/\text{poly}(n)$	[19]
$O(\log^2(n)\log(1/\delta_2))$	$p = 0$	perfect L_0 sampler	[35]
$\text{poly}\log(\nu^{-1}, n)$	$p \in [0, 2]$	$\delta = 1/\text{poly}(n)$	[45]
$O(\nu^{-p}\log^3(n)\log(1/\delta))$	$p \in [1, 2]$	$(1 \pm \nu)$-relative error	[1]
$O(\nu^{-\max\{1,p\}}\log^2(n)\log(1/\delta))$	$p \in (0, 2) \setminus \{1\}$	$(1 \pm \nu)$-relative error	[35]
$O(\nu^{-1}\log(\nu^{-1})\log^2(n)\log(1/\delta))$	$p = 1$	$(1 \pm \nu)$-relative error	[35]
$O(\log^2(n)\log(1/\delta))$	$p \in (0, 2)$	perfect L_p sampler, matches lower bound	This work
$O(\log^3(n)\log(1/\delta))$	$p = 2$	perfect L_2 sampler	This work

Fig. 1: Evolution of one pass L_p sampling upper bounds, with the best known lower bound of $\Omega(\log^2(n)\log(1/\delta))$ for $p \geq 0$ [37] (see also [35] for a lower bound for constant δ).

The one-pass approximate L_p sampler introduced in [45] requires $\text{poly}(\nu^{-1}, \log n)$ space, albeit with rather large exponents. Later on, in [1], the complexity was reduced significantly to $O(\nu^{-p}\log^3(n)\log(1/\delta))$-bits[1] for $p \in [1, 2]$, using a technique known as *precision sampling*. Roughly, the technique of precision sampling consists of scaling the coordinates f_i by random variable coefficients $1/t_i$ as the updates arrive, resulting in a new stream vector $z \in \mathbb{R}^n$ with $z_i = f_i/t_i$. The algorithm then searches for all z_i which cross a certain threshold T. Observe that if $t_i = u_i^{1/p}$ where u_i is uniform on $[0, 1]$, then the probability that $f_i/t_i \geq T$ is precisely $\Pr[u_i < |f_i|^p/T^p] = |f_i|^p/T^p$. By running an L_p estimation algorithm to obtain $T \in [\frac{1}{2}\|f\|_p, \frac{3}{2}\|f\|_p]$, an L_p sampler can then return any i with $z_i \geq T$ as its output. These heavy coordinates can be found using any of the well-known η-heavy hitters algorithms for a sufficiently small precision η.

Using a tighter analysis of this technique with the same scaling variables $t_i = u_i^{1/p}$, Jowhari, Sağlam, and Tardos reduced the space complexity of L_p sampling for $p < 2$ to $O(\nu^{-\max\{1,p\}}\log^2(n)\log(1/\delta))$-bits for $p \in (0, 2) \setminus \{1\}$, and $O(\nu^{-1}\log(\nu^{-1})\log^2(n)\log(1/\delta))$ bits of space for $p = 1$ [35]. Roughly speaking, their improvements result from a more careful consideration of the precision η needed to determine when a z_i crosses the threshold, which they do via the tighter tail-error guarantee of the well-known count-sketch heavy hitters algorithm [6]. In addition, they give an $O(\log^2(n)\log(1/\delta))$ perfect L_0 sampler, and demonstrated an $\Omega(\log^2(n))$-bit lower bound for L_p samplers for any $p \geq 0$. Recently, this lower bound was extended to $\Omega(\log^2(n)\log(1/\delta))$ [37] bits, which closes the complexity of the problem for $p = 0$.

For $p \in (0, 2)$, this means that the upper and lower bounds for L_p samplers are tight in terms of n, δ, but a gap exists in the dependency on ν. Being the case, it would seem natural to search for an $\Omega(\nu^{-p}\log^2(n)\log(1/\delta))$ lower bound to close the complexity of the problem. It is perhaps surprising, therefore, that no lower bound at all in terms of ν exists – not even an $\Omega(\nu^{-1})$ bound is known. This poses the question of whether the $\Omega(\log^2(n)\log(1/\delta))$ lower bound is in fact correct.

[1]We note that previous works [35], [37] have cited the sampler of [1] as using $O(\log^3(n))$-bits of space, however the space bound given in their paper is in *machine words*, and is therefore an $O(\log^4(n))$ bit bound with $\delta = 1/\text{poly}(n)$. In order to obtain an $O(\log^3(n)\log(1/\delta))$ bit sampler, their algorithm must be modified to use fewer repetitions.

A. Our Contributions

In this paper, we explain the phenomenon of the lack of an $\Omega(\nu^{-1})$ lower bound by showing that ν need not enter the space complexity of an L_p sampler at all. In other words, we demonstrate the existence of *perfect L_p samplers* using $O(\log^2(n)\log(1/\delta))$-bits of space for $p \in (0, 2)$, thus completely resolving the space complexity of the problem. For $p = 2$, our space is $O(\log^3(n)\log(1/\delta))$-bits, which matches the best known upper bounds in terms of n, δ, yet again has no dependency on ν. A summary of the past upper bounds for L_p sampling, along with the contributions of this work, is given in Figure 1.

In addition to outputting a perfect sample i from the stream, for $p \in (0, 2)$ we also show that, conditioned on an index being output, given an additional additive $O(\min\{\epsilon^{-2}, \max\{\epsilon^{-p}, \log(\frac{1}{\delta_2})\}\}\log(n)\log(1/\delta_2))$-bits we can provide a $(1 \pm \epsilon)$ approximation of the frequency $|f_i|$ with probability $1 - \delta_2$. This separates the space dependence on $\log^2(n)$ and ϵ for frequency approximation, allowing us to obtain a $(1 \pm \epsilon)$ approximation of $|f_i|$ in $O(\log^2(n) + \epsilon^{-p}\log(n))$ bits of space with constant probability, whereas before this required $O(\epsilon^{-p}\log^2(n))$ bits of space. For $p = 2$, our bound is $O(\epsilon^{-2}\log^2(n)\log(1/\delta_2))$, which still improves upon the prior best known bounds for estimating the frequency by an $O(\log(n))$-factor. Finally, we show an $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space lower bound for producing the $(1 \pm \epsilon)$ estimate (conditioned on an index being returned), which is tight for most ranges of ϵ.

B. Applications

Since their introduction, it has been observed that L_p samplers can be used as a building block in algorithms for many important streaming problems, such as finding heavy hitters, L_p-norm estimation, cascaded norm estimation, and finding duplicates in data streams [1], [45], [35], [5]. L_p

Input: $f \in \mathbb{R}^n$
Output: a sampled index $i^* \in [n]$
1) Perform a linear transformation on f to obtain z.
2) Run instance A of count-sketch on z to obtain the estimate y.
3) Find $i^* = \arg\max_i |y_i|$. Then run a statistical test on y to decide whether to output i^* or FAIL.

Fig. 2: Algorithmic Template for L_p sampling

samplers, particularly for $p = 1$, are often used as a black-box subroutine to design representative histograms of f on which more complicated algorithms are run [25], [24], [49], [27], [31], [13]. For these black-box applications, the only property of the samplers needed is the distribution of their samples. Samplers with relative error are statistically biased and, in the analysis of more complicated algorithms built upon such samplers, this bias and its propagation over multiple samples must be accounted for and bounded. The analysis and development of such algorithms would be simplified dramatically, therefore, with the assumptions that the samples were truly uniform (i.e., from a perfect L_1 sampler). In this case, no error terms or variational distance need be accounted for. Our results show that such an assumption is possible without affecting the space complexity of the sampler.

Note that in Definition 1, we allow a perfect sampler to have n^{-c} variation distance to the true L_p distribution. We note that this definition is in line with prior work, observing that even the perfect L_0 sampler of [35] incurs such an error from derandomizing with Nisan's PRG. Nevertheless, this error will never be detected if the sampler is run polynomially many times in the course of constructing a histogram, and such a sampler is therefore statistically indistinguishable from a truly uniform sampler and can be used as a black box.

Another motivation for utilizing perfect L_p samplers comes from applications in privacy. Here $f \in \mathbb{R}^n$ is some underlying dataset, and we would like to reveal a sample $i \in [n]$ drawn from the L_p distribution over f to some external party without revealing too much global information about f itself. Using an approximate L_p sampler introduces a $(1 \pm \nu)$ multiplicative bias into the sampling probabilities, and this bias can depend on global properties of the data. For instance, such a sampler might bias the sampling probabilities of a large set S of coordinates by a $(1 + \nu)$ factor if a certain global property P holds for f, and may instead bias them by $(1 - \nu)$ if a disjoint property P' holds. Using only a small number of samples, an adversary would then be able to distinguish whether P or P' holds by determining how these coordinates were biased. On the other hand, the bias in the samples produced by a perfect L_p sampler is polynomially small, and thus the leakage of global information could be substantially smaller when using one, though one would need to formally define a notion of leakage and privacy for the given application.

C. Our Techniques

Our main algorithm is inspired by the precision sampling technique used in prior works [1], [35], but with some marked differences. To describe how our sampler achieves the improvements cited above, we begin by observing that all L_p sampling algorithms since [1] have adhered to the same algorithmic template, shown in Figure 2. This template employs the classic *count-sketch* algorithm of [7] as a subroutine, which is easily introduced. For $k \in \mathbb{N}$, let $[k]$ denote the set $\{1, 2, \ldots, k\}$. Given a precision parameter η, count-sketch selects pairwise independent hash functions $h_j : [n] \to [6/\eta^2]$ and $g_j : [n] \to \{1, -1\}$, for $j = 1, 2, \ldots, d$ where $d = \Theta(\log(n))$. Then for all $i \in [d]$, $j \in [6/\eta^2]$, it computes the following linear function $A_{i,j} = \sum_{k \in [n], h_i(k) = j} g_i(k) f_k$, and outputs an approximation y of f given by $y_k = \text{median}_{i \in [d]}\{g_i(k) A_{i,h_i(k)}\}$. We will discuss the estimation guarantee of count-sketch at a later point.

The algorithmic template is as follows. First, perform some linear transformation on the input vector f to obtain a new vector z. Next, run an instance A of count-sketch on z to obtain the estimate y. Finally, run some statistical test on y. If the test fails, then output FAIL, otherwise output the index of the largest coordinate (in magnitude) of y. We first describe how the sampler of [35] implements the steps in this template. Afterwards we describe the different implementation decisions made in our algorithm that allow it to overcome the limitations of prior approaches.

Prior Algorithms.: The samplers of [35], [1] utilize the technique known as *precision sampling*, which employs the following linear transformation. The algorithms first generate random variables $(t_1, \ldots, t_n)$ with limited independence, where each $t_i \sim \text{Uniform}[0, 1]$. Next, each coordinate f_i is scaled by the coefficient $1/t_i^{1/p}$ to obtain the transformed vector $z \in \mathbb{R}^n$ given by $z_i = f_i / t_i^{1/p}$, thus completing Step 1 of Figure 2. For simplicity, we now restrict to the case of $p = 1$ and the algorithm of [35]. The goal of the algorithm is then to return an item z_i that crosses the threshold $|z_i| > \nu^{-1} R$, where $R = \Theta(\|f\|_1)$ is a constant factor approximation of the L_1. Note the probability that this occurs is proportional to $\nu |f_i| / \|f\|_1$.

Next, implementing the second step of Figure 2, the vector z is hashed into count-sketch to find an item that has crossed the threshold. Using the stronger *tail-guarantee* of count-sketch, the estimate vector y satisfies $\|y - z\|_\infty \leq \sqrt{\eta} \|z_{\text{tail}(1/\eta)}\|_2$, where $z_{\text{tail}(1/\eta)}$ is z with the $1/\eta$ largest coordinates (in magnitude) set to 0. Now the algorithm runs into trouble when it incorrectly identifies z_i as crossing the threshold when it has not, or vice-versa. However, if the tail error $\sqrt{\eta} \|z_{\text{tail}(1/\eta)}\|_2$ is at most $O(\|f\|_1)$, then since t_i is a uniform variable the

probability that z_i is close enough to the threshold to be misidentified is $O(\nu)$, which results in at most $(1 \pm \nu)$ relative error in the sampling probabilities. Thus it will suffice to have $\sqrt{\eta}\|z_{\text{tail}(1/\eta)}\|_2 = O(\|f\|_1)$ with probability $1 - \nu$. To show that this is the case, consider the level sets $I_k = \{z_i \mid z_i \in (\frac{\|f\|_p}{2^{(k+1)/p}}, \frac{\|f\|_p}{2^{k/p}})\}$, and note $\mathbb{E}[|I_k|] = 2^k$. We observe here that results of [35] can be partially attributed to the fact that for $p < 2$, the total contribution $\Theta(\frac{\|f\|_p^2}{2^{2k/p}}|I_k|)$ of the level sets to $\|z\|_2^2$ decreases geometrically with k, and so with constant probability we have $\|z\|_2 = O(\|f\|_p)$. Moreover, if one removes the top $\log(1/\nu)$ largest items, the contribution of the remaining items to the L_2 is $O(\|f\|_1)$ with probability $1-\nu$. So taking $\eta = \log(1/\nu)$, the tail error from count-sketch has the desired size. Since the tail error does not include the $1/\eta$ largest coordinates, this holds even conditioned on a fixed value t_{i^*} of the maximizer.

Now with probability ν the guarantee on the error from the prior paragraph does not hold, and in this case one *cannot* still output an index i, as this would result in a *ν-additive* error sampler. Thus, as in Step 3 of Figure 2, the algorithm must implement a statistical test to check that the guarantee holds. To do this, using the values of the largest $1/\eta$ coordinates of y, they produce an estimate of the tail-error and output FAIL if it is too large. Otherwise, the item $i^* = \arg\max_i |y_i|$ is output if $|y_{i^*}| > \nu^{-1}R$. The whole algorithm is run $O(\nu^{-1}\log(1/\delta))$ times so that an index is output with probability $1 - \delta$.

Our Algorithm.: Our first observation is that, in order to obtain a truly perfect sampler, one needs to use different scaling variables t_i. Notice that the approach of scaling by inverse uniform variables and returning a coordinate which reaches a certain threshold T faces the obvious issue of what to return when more than one of the variables $|z_i|$ crosses T. This is solved by simply outputting the maximum of all such coordinates. However, the probability of an index becoming the maximum *and* reaching a threshold is drawn from an entirely different distribution, and for uniform variables t_i this distribution does not appear to be the correct one. To overcome this, we must use a distribution where the maximum index i of the variables $(|f_1 t_2^{-1/p}|, |f_2 t_2^{-1/p}|, \ldots, |f_n t_n^{-1/p}|)$ is drawn *exactly* according to the L_p distribution $|f_i|^p/\|f\|_p^p$. We observe that the distribution of exponential random variables has precisely this property, and thus to implement Step 1 of Figure 2 we set $z_i = f_i/t_i^{1/p}$ where t_i is an exponential random variable. We remark that exponential variables have been used in the past, such as for F_p moment estimation, $p > 2$, in [1] and regression in [54]. However it appears that their applicability to sampling has never before been exploited.

Next, we carry out the count-sketch step by hashing our vector z into A with sensitivity parameter η. Using a more general analysis of the L_2 norm of the level sets I_k, we can show that, with constant probability, the error in estimating the frequency of an item z_i from our count-sketch is $O(\|f\|_p)$. Then by applying the *min-stability* property of exponential random variables, we can show that the maximum item $|z_{(1)}| = \max\{|z_i|\}$ is distributed as $\|f\|_p/E^{1/p}$, where E is another exponential random variable. Thus $|z_{(1)}| = O(\|f\|_p)$ with constant probability, and we can similarly show that $|z_{(1)}| - |z_{(2)}| = O(\|f\|_p)$ with constant probability, where $|z_{(2)}|$ is the second largest coordinate (in absolute value) of z. If all these events occur together, we will be able to distinguish the largest coordinate in z from the second largest, thus finding the coordinate i with $z_i = z_{(1)}$. However, just as in [35], we cannot output an index anyway if these conditions do not hold, so we will need to run a statistical test to ensure that they do.

The Statistical Test.: Let R be a constant factor approximation of $\|f\|_p$. To implement Step 3 of the template, our algorithm tests whether $|y_{(1)}| - |y_{(2)}| \geq q\eta R$, for some sufficiently large constant q (where $y_{(1)}$ and $y_{(2)}$ are the largest and second largest coordinates, in magnitude, of y). The primary technical challenge will be to show that, conditioned on $|z_i| = z_{(1)}$ for some i, the probability of failing the statistical test *does not depend on* i. In other words, conditioning on $|z_i|$ being the maximum does not change the failure probability. Let $z_{D(k)}$ be the k-th order statistic of z (i.e., $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$). Here the $D(k)$'s are known as *anti-ranks*. To analyze the conditional dependence, we must first obtain a closed form for $z_{D(k)}$ which separates the dependencies on k and $D(k)$. Hypothetically, if $z_{D(k)}$ depended only on k, then a test of the form $z_{D(1)} - z_{D(2)} < T$, for some threshold T, would be completely independent of $D(1)$, in which case we could safely fail whenever such an event occurred. Of course, in reality this is not the case. Consider the vector $f = (100n, 1, 1, 1, \ldots, 1) \in \mathbb{R}^n$ and $p = 1$. Clearly we expect z_1 to be the maximizer, and moreover we expect a gap of $\Theta(n)$ between z_1 and $z_{D(2)}$. On the other hand, if you were told that $D(1) \neq 1$, it is tempting to think that $z_{D(1)}$ just *barely* beat out z_1 for its spot as the max, and so z_1 would not be far behind. Indeed, this intuition would be correct, and one can show that the probability that $z_{D(1)} - z_{D(2)} > n$ conditioned on $D(1) = i$ changes by an additive constant depending on whether or not $i = 1$. In general, the value of $|z_{D(2)}|$ changes by a $\|f\|_p^p/\|f_{-D(1)}\|_p^p$ multiplicative factor, where $f_{-D(1)}$ is f with the coordinate $D(1)$ removed.

To handle scenarios of this form, our algorithm will utilize an additional linear transformation in Step 1 of the template. Instead of only scaling by the random coefficients $1/t_i^{1/p}$, our algorithm first *duplicates* the coordinates f_i to remove all heavy items from the stream. If f is the vector from the example above and F is the duplicated vector, then after $\text{poly}(n)$ duplications all copies of the heavy item f_1 will have weight at most $|f_1|/\|F\|_1 < 1/\text{poly}(n)$. By uniformizing the relative weight of the coordinates, this washes out the dependency of $|z_{D(2)}|$ on $D(1)$, since $\|F_{-D(1)}\|_p^p = (1 \pm n^{-\Omega(c)})\|F_{-j}\|_p^p$ after n^c duplications, for any $j \in [n^c]$. Notice that this transformation blows up the dimension of f by a $\text{poly}(n)$ factor. However, since our space usage is $O(\log^2 n)$-bits, the result is only a constant factor increase in the complexity.

After duplication, we scale F by the coefficients $1/t_i^{1/p}$, and the rest of the algorithm proceeds as described above. Using expressions for the order statistics $z_{D(k)}$ which separate the

dependence into the anti-ranks $D(j)$ and a set of exponentials $E_1, E_2, \dots E_n$ *independent* of the anti-ranks, after duplication we can derive tight concentration of the $z_{D(k)}$'s conditioned on fixed values of the E_i's. Using this concentration result, we decompose our count-sketch data structure A into two component variables: one independent of the anti-ranks (the independent component), and a small adversarial noise of relative weight n^{-c}. In order to bound the effect of the adversarial noise on the outcome of our tests we must $\mathbf{1}$) randomize the threshold for our failure condition and $\mathbf{2}$) demonstrate the anti-concentration of the resulting distribution over the independent components of A. This will demonstrate that with high probability, the result of the statistical test is completely determined by the value of the independent component, which allows us to fail without affecting the conditional probability of outputting $i \in [n]$.

Now the correctness of our sampler crucially relies on the full independence of the t_i's to show that the variable $D(1)$ is drawn from precisely the correct distribution (namely, the L_p distribution $|f_i|^p/\|f\|_p^p$). Being the case, we cannot directly implement our algorithm using any method of limited independence. In order to derandomize the algorithm from requiring full-independence, we will use Nisan's pseudorandom generator [48]. Using Nisan's generator as a blackbox, however, results in an $O(\log n)$-blow up in the space complexity of an algorithm. In order to avoid this, we carry out a closer analysis of the exact seed length that Nisan's generator requires to fool a statistical tester of our algorithm, showing that $O(\log^2 n)$-bits suffices and therefore leaving our overall space complexity unchanged.

Case of $p = 2$.: Recall for $p < 2$, we could show that the L_2 of the level sets I_k decays geometrically with k. More precisely, for any γ we have $\|z_{\text{tail}(\gamma)}\|_2 = O(\|F\|_p \gamma^{-1/p+1/2})$ with probability $1 - O(e^{-\gamma})$. Using this, we actually do not need the tight concentration of the $z_{D(k)}$'s, since we can show that the top $n^{c/10}$ coordinates change by at most $(1 \pm n^{-\Omega(c)})$ depending on $D(1)$, and the L_2 norm of the remaining coordinates is only an $O(n^{-c/10(1/p-1/2)})$ fraction of the whole L_2, and can thus be absorbed into the adversarial noise. For $p = 2$ however, each level set I_k contributes weight $O(\|F\|_p^2)$ to $\|z\|_2^2$, so $\|z_{\text{tail}(\gamma)}\|_2 = O(\sqrt{\log(n)}\|F\|_p)$ even for $\gamma = \text{poly}(n)$. Therefore, for $p = 2$ it is essential that we show concentration of the $z_{D(k)}$'s for *nearly all* k. Moreover, the error from count-sketch will be $O(\eta\sqrt{\log(n)}\|F\|_p)$ with high probability, and for any L with probability $1 - 1/L$ we have $|z_{D(1)}| < \sqrt{L}\|F\|_p$. Therefore, the only way to find the maximum index with constant probability will be to take $\eta = \Theta(1/\sqrt{\log(n)})$ in our count-sketch, which can be accomplished with $O(\log^3 n)$-bits of space.

Optimizing the Runtime.: In addition to our core sampling algorithm, we show how the linear transformation step to construct z can be implemented via a parameterized rounding scheme to improve the update time of the algorithm without affecting the space complexity, giving a run-time/relative sampling error trade-off. By rounding the scaling variables $1/t_i^{1/p}$ to powers of $(1 + \nu)$, we discretize their support to have size $O(\nu \log(n))$. We then simulate the update procedure by sampling from the distribution over updates to our count-sketch data-structure A of duplicating an update and hashing each duplicate independently into A. Our simulation utilizes results on efficient generation of binomial random variables, through which we can iteratively reconstruct the updates to A bin-by-bin instead of duplicate-by-duplicate. In addition, by using an auxiliary heavy-hitter data structure, we can improve our query time from the naïve $O(n)$ to $O(\text{poly}\log(n))$ without increasing the space complexity.

Estimating the Frequency.: We show that allowing an additional additive $O(\min\{\epsilon^{-2}, \max\{\epsilon^{-p}, \log(\frac{1}{\delta_2})\}\}\log n \log \delta_2^{-1})$ bits of space, we can provide an estimate $\tilde{f} = (1 \pm \epsilon)f_i$ of the outputted frequency f_i with probability $1 - \delta_2$ when $p < 2$. To achieve this, we use our more general analysis of the contribution of the level sets I_k to $\|z\|_2$, and give concentration bounds on the tail error when the top ϵ^{-p} items are removed. When $p = 2$, for similar reasons as described in the sampling algorithm, we require another $O(\log n)$ factor in the space complexity to obtain a $(1 \pm \epsilon)$ estimate. Finally, we demonstrate an $\Omega(\epsilon^{-p} \log n \log \delta_2^{-1})$ lower bound for this problem, which is tight when $p < 2$ for most values of ϵ. To do so, we adapt a communication problem introduced in [34], known as *Augmented-Indexing on Large Domains*. We weaken the problem so that it need only succeed with constant probability, and then show that the same lower bound still holds. Using a reduction to this problem, we show that our lower bound for L_p samplers holds even if the output index is from a distribution with constant *additive* error from the true L_p distribution $|f_i|^p/\|f\|_p^p$.

II. Preliminaries

For $a, b, \epsilon \in \mathbb{R}$, we write $a = b \pm \epsilon$ to denote the containment $a \in [b - \epsilon, b + \epsilon]$. For positive integer n, we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. For any vector $v \in \mathbb{R}^n$, we write $v_{(k)}$ to denote the k-th largest coordinate of v in absolute value. In other words, $|v_{(1)}| \geq |v_{(2)}| \geq \cdots \geq |v_{(n)}|$. For any $\gamma \in [n]$, we define $v_{\text{tail}(\gamma)}$ to be v but with the top γ coordinates (in absolute value) set equal to 0. We write $|v|$ to denote the entry-wise absolute value of v, so $|v|_j = |v_j|$ for all $j \in [n]$. All space bounds stated will be in bits. For our runtime complexity, we assume the unig cost RAM model, where a word of $O(\log(n))$-bits can be operated on in constant time, where n is the dimension of the input streaming vector. Finally, we will use the $\tilde{O}$ notation to hide $\text{polylog}(n)$ factors, in other words $O(\log^c(n)) = \tilde{O}(1)$ for any constant c.

Formally, a data stream is given by an underlying vector $f \in \mathbb{R}^n$, called the *frequency vector*, which is initialized to 0^n. The frequency vector then receives a stream of m updates of the form $(i_t, \Delta_t) \in [n] \times \{-M, \dots, M\}$ for some $M > 0$ and $t \in [m]$. The update (i, Δ) causes the change $f_{i_t} \leftarrow f_{i_t} + \Delta_t$. For simplicity, we make the common assumption ([4]) that $\log(mM) = O(\log(n))$, though our results generalize naturally to arbitrary n, m. In this paper, we will also need the McDiarmid inequality.

Fact 1 (McDiarmid's inequality [44]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables, and let $\psi(x_1, \ldots, x_n)$ by any function that satisfies $\sup_{x_1,\ldots,x_n,\hat{x}_i} \left|\psi(x_1, x_2, \ldots, x_n) - \psi(x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_n)\right| \leq c_i$ for $1 \leq i \leq n$. Then for any $\epsilon > 0$, we have $\Pr\left[\left|\psi(X_1, \ldots, X_n) - \mathbb{E}\left[\psi(X_1, \ldots, X_n)\right]\right| \geq \epsilon\right] \leq 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i}\right)$.*

A. Count-Sketch

Our sampling algorithm will utilize the well known data structure known as *count-sketch*(see [7] for further details). We now introduce the description of count-sketch which we will use for the remainder of the paper. The count-sketch data structure is a table A with d rows and k columns. When run on a vector $f \in \mathbb{R}^n$ presented in a stream, for each row $r \in [d]$, count-sketch picks a uniform random mapping $h_i : [n] \to [k]$ and $g_i : [n] \to \{1, -1\}$. Generally, h_i and g_i need only be 4-wise independent hash functions, but in this paper we will use fully-independent hash functions (and later relax this condition when derandomizing). Whenever an update Δ to item $v \in [n]$ occurs, count-sketch performs the following updates:

$$A_{i,h_i(v)} \leftarrow A_{i,h_i(v)} + \Delta g_i(v) \quad \text{for } i = 1, 2, \ldots, d$$

Note that while we will not implement the h_i's as explicit hash functions, and instead generate i.i.d. random variables $h_i(1), \ldots, h_i(n)$, we will still use the terminology of hash functions. In other words, by *hashing* the update (v, Δ) into the row A_i of count-sketch, we mean that we are updating $A_{i,h_i(v)}$ by $\Delta g_i(v)$. By hashing the coordinate f_v into A, we mean updating $A_{i,h_i(v)}$ by $g_i(v)f_v$ for each $i = 1, 2, \ldots, d$. Using this terminology, each row of count-sketch corresponds to randomly hashing the indices in $[n]$ into k buckets, and then each bucket in the row is a sum of the frequencies f_i of the items which hashed to it multiplied by random ± 1 signs. At the termination of the stream, we define a vector y as: $y_j = \text{median}_{i \in [d]} A_{i,h_i(j)} g_i(j)$.

Theorem 1. *If $d = \Theta(\log(1/\delta))$ and $k = 6/\epsilon^2$, then for a fixed $i \in [n]$ we have $|y_i - f_i| < \epsilon \|f_{tail(1/\epsilon^2)}\|_2$ with probability $1 - \delta$. Moreover, if $d = \Theta(\log(n))$ and $c \geq 1$ is any constant, then we have $\|y - f\|_\infty < \epsilon \|f_{tail(1/\epsilon^2)}\|_2$ with probability $1 - n^{-c}$. Furthermore, if we instead set $y_j = median_{i \in [d]} |A_{i,h_i(j)}|$, then the same two bounds above hold replacing f with $|f|$.*

B. High Probability L_p estimation

Since our later proofs will rely on a precise analysis of the random variables which are added into our data structures, we will review the specifics of the L_2 estimation algorithm we will be using. The algorithm uses the well-known p-stable classes of distributions. We give a simplified definition here.

Definition 2. A distribution $\mathcal{D}_p$ is said to be p-stable if whenever $X_1, \ldots, X_n \sim \mathcal{D}_p$ are drawn independently, we have

$$\sum_{i=1}^n a_i X_i = \|a\|_p X$$

for any fixed vector $a \in \mathbb{R}^n$, where $X \sim \mathcal{D}_p$ is again distributed as a p-stable. In particular, the Gaussian random variables $\mathcal{N}(0, 1)$ are p-stable for $p = 2$ (i.e., $\sum_i a_i g_i = \|a\|_g$ where $g, g_1, \ldots, g_n$ are Gaussian).

These distributions were originally utilized by [32] to derive $O(\log^2(n)))$-space estimators for the L_p norm for $p \in (0, 2]$. The space complexity came from storing random bits to generate $O(n)$ p-stable random variables, and then derandomizing the entire algorithm via Nisan's PRG [48]. This algorithm was later refined in [36] to use $O(\log(n))$-bits of space by showing that variables with limited independence suffice to obtain an estimate. Since the algorithms in this paper already use $O(\log^2(n))$-bits and we only need a constant factor approximation, we will use the median estimator of [32] for simplicity.

The algorithm is as follows. To estimate $\|f\|_2$ for $f \in \mathbb{R}^n$, we generate i.i.d. Gaussians $\varphi_{i,j} \sim \mathcal{N}(0, 1)$ for $i \in [n]$ and $j \in [r]$ where $r = \Theta(\log(n))$. Note that we will later derandomize this procedure with Nisan's PRG to generate the $\varphi_{i,j}$'s using a random seed of length $O(\log^2(n))$. We then store the vector $B \in \mathbb{R}^r$ where $B_j = \sum_{i=1}^n f_i \varphi_{i,j}$ for $j = 1, \ldots, r$, which can be computed update by update throughout the stream. We then return the estimate $R = \text{median}_j \frac{5|B_j|}{4}$.

Lemma 1. *For any constant $c > 0$, The value of R as computed in the above algorithm satisfies $\frac{1}{2}\|f\|_2 \leq R \leq 2\|f\|_2$ with probability $1 - n^{-c}$.*

Proof. Each coordinate B_j is distributed as $|B_j| = |g_j| \|f\|_2$, where g_j are i.i.d. Gaussian random variables. A simple computation shows that $\Pr[|g_j| \in [2/5, 8/5]] > .55$, and thus $\Pr[(5/4)|B_j| \in [1/2\|f\|_2, 2\|f\|_2]] > .55$. Then by Chernoff-Hoeffding bounds, the median of $O(\log(n))$ repetitions satisfies this bound with probability $1 - n^{-c}$ as stated. □

Finally, we will use the fact that high probability L_p estimators can be obtained in $O(\log^2(n))$ space for $p < 2$. Such an algorithm can be found in [36].

Lemma 2 ([36]). *For $p \in (0, 2)$ and any constant c, given a general turnstile stream f, there is an algorithm which produces an estimate $R' \in [\frac{1}{2}\|f\|_p, 2\|f\|_p]$ with probability $1 - n^{-c}$. The space required is $O(\log^2(n))$ bits, and the update and query time are both $O(1)$ in the unit cost RAM model.*

III. Exponential Order Statistics

In this section, we discuss several useful properties of the order statistics of n independent non-identically distributed exponential random variables. Let $(t_1, \ldots, t_n)$ be independent exponential random variables where t_i has mean $1/\lambda_i$ (equivalently, t_i has rate λ_i). Recall that t_i is given by the cumulative distribution function $\Pr[t_i < x] = 1 - e^{-\lambda_i x}$. We begin by noting that constant factor scalings of an exponential variable result in another exponential variable.

Fact 2 (Scaling of exponentials). *Let t be exponentially distributed with rate λ, and let $\alpha > 0$. Then αt is exponentially distributed with rate λ/α*

Proof. The cdf of αt is given by $\Pr[t < x/\alpha] = 1 - e^{-\lambda x/\alpha}$, which is the cdf of an exponential random variable with rate λ/α. □

We would now like to study the order statistics of the variables $(t_1, \ldots, t_n)$, where t_i has rate λ_i. To do so, we introduce the *anti-rank vector* $(D(1), D(2), \ldots, D(n))$, where for $k \in [n]$, $D(k) \in [n]$ is a random variable which gives the index of the t-th smallest exponential.

Definition 3. Let $(t_1, \ldots, t_n)$ be independent exponentials. For $k = 1, 2, \ldots, n$, we define the *k-th anti-rank* $D(k) \in [n]$ of $(t_1, \ldots, t_n)$ to be the values $D(k)$ such that $t_{D(1)} \leq t_{D(2)} \leq \cdots \leq t_{D(n)}$.

Using the structure of the anti-rank vector, it has been observed [47] that there is a simple form for describing the distribution of $t_{D(k)}$ as a function of $(\lambda_1, \ldots, \lambda_n)$ and the anti-rank vector.

Fact 3 ([47])**.** *Let $(t_1, \ldots, t_n)$ be independently distributed exponentials, where t_i has rate $\lambda_i > 0$. Then for any $k = 1, 2, \ldots, n$, we have*

$$t_{D(k)} = \sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{n} \lambda_{D(j)}}$$

Where the $E_1, E_2, \ldots, E_n$'s are i.i.d. exponential variables with mean 1 and are independent of the anti-rank vector $(D(1), D(2), \ldots, D(n))$.

Fact 4 ([47])**.** *For any $i = 1, 2, \ldots, n$, we have*

$$\Pr[D(1) = i] = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

We now describe how these properties will be useful to our sampler. Let $f \in \mathbb{R}^n$ be any vector of a general turnstile stream. We can generate i.i.d. exponentials $(t_1, \ldots, t_n)$, each with rate 1, and construct the random variable $z_i = f_i / t_i^{1/p}$, which can be obtained in a stream by scaling updates to f_i by $1/t_i^{1/p}$ as they arrive. By Fact 2, the variable $|z_i|^{-p} = t_i / |f_i|^p$ is exponentially distributed with rate $\lambda_i = |f_i|^p$. Now let $(D(1), \ldots, D(n))$ be the anti-rank vector of the exponentials $(t_1/|f_n|^p, \ldots, t_n/|f_n|^{1/p})$. By Fact 4, we have $\Pr[D(1) = i] = \Pr[i = \arg\min\{|z_1|^{-p}, \ldots, |z_n|^{-p}\}] = \Pr[i = \arg\max\{|z_1|, \ldots, |z_n|\}] = \frac{\lambda_i}{\sum_j \lambda_j} = \frac{|f_i|^p}{\|f\|_p^p}$. In other words, the probability that $|z_i| = \arg\max_j\{|z_j|\}$ is precisely $|f_i|^p / \|f\|_p^p$, so for a perfect L_p sampler it suffices to return $i \in [n]$ with $|z_i|$ maximum.

Now note $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$, and in this scenario the statement of Fact 3 becomes:

$$z_{D(k)} = \Big(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} \lambda_{D(j)}}\Big)^{-1/p} = \Big(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} F_{D(j)}^p}\Big)^{-1/p}$$

Where the E_i's are i.i.d. exponential random variables with mean 1, and are independent of the anti-rank vector $(D(1), \ldots, D(n))$. We call the exponentials E_i the *hidden exponentials*, as they do not appear in the actual execution of the algorithm, and will be needed for analysis purposes only.

IV. The Sampling Algorithm

We now provide intuition for the workings of our main sampling algorithm. Our algorithm scales the input stream by inverse exponentials to obtain a new vector z. We have seen in the prior section that we can write the order statistics $z_{D(k)}$ as a function of the anti-rank vector D, where $D(k)$ gives the index of the k-th largest coordinate in z, and the hidden exponentials E_i, which describe the "scale" of the order statistics. Importantly, the hidden exponentials are independent of the anti-ranks. We would like to determine the index i such that $D(1) = i$, however this may not always be possible. This is the case when the two largest elements $|z_{D(1)}|, |z_{D(2)}|$ are closer to each other than our error in estimating them. In such a case, we would like to output `FAIL`.

Now consider the trivial situation where $f_1 = f_2 = \cdots = f_n$. Here the variables $z_{D(k)}$ have no dependence at all on the anti-rank vector D. In this case, the condition of failing is independent of $D(1)$, so we can safely fail whenever we cannot determine the maximum index. On the other hand, if the values $|f_i|$ vary wildly, the variables $z_{D(k)}$ will depend highly on the anti-ranks. In fact, if there exists f_i with $|f_i|^p \geq \epsilon \|f\|_p^p$, then the probability that $|z_{D(1)}| - |z_{D(2)}|$ is above a certain threshold can change by a $(1 \pm \epsilon)$ factor conditioned on $D(1) = i$, as opposed to $D(1) = j$ for a smaller $|f_j|$. In this case, we cannot output `FAIL`, lest we suffer a $(1 \pm \epsilon)$ error in outputting an index with the correct probability.

To handle this, we must remove the heavy items from the stream to weaken the dependence of the values $z_{D(k)}$ on the anti-ranks, which we carry out by duplication of coordinates. For the purposes of efficiency, we carry out the duplication via a rounding scheme which will allow us to generate and quickly hash updates into our data-structures (Section V). We will show that, conditioned on the fixed values of the E_i's, the variables $z_{D(k)}$ are highly concentrated, and therefore nearly independent of the anti-ranks ($z_{D(k)}$ depends only on k and not $D(k)$). By randomizing the failure threshold to be anti-concentrated, the small adversarial dependence of $z_{D(k)}$ on $D(k)$ cannot non-trivially affect the conditional probabilities of failure, leading to small relative error in the resulting output distribution.

The L_p Sampler.: We now describe our sampling algorithm, as given in Figure 3. Let $f \in \mathbb{R}^n$ be the input vector of the stream. As the stream arrives, we duplicate updates to each coordinate f_i n^{c-1} times to obtain a new vector $F \in \mathbb{R}^{n^c}$. More precisely, for $i \in [n]$ we set $i_j = (i-1)n^{c-1} + j$ for $j = 1, 2, \ldots, n^{c-1}$, and then we will have $F_{i_j} = f_i$ for all $i \in [n]$ and $j \in [n^{c-1}]$. We then call F_{i_j} a duplicate of f_i. Whenever we use i_j as a subscript in this way it will refer to a duplicate of i, whereas a single subscript i will be used both to index into $[n]$ and $[n^c]$. Note that this duplication has the effect that $|F_i|^p \leq n^{-c+1}\|F\|_p^p$ for all $p > 0$ and $i \in [n^c]$.

We then generate *i.i.d.* exponential rate 1 random variables $(t_1, \ldots, t_n)$, and define the vector $z \in \mathbb{R}^{n^c}$ by $z_i = F_i / t_i^{1/p}$. As shown in Section III, we have $\Pr[i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}] = |F_{i_j}|^p / \|F\|_p^p$. Since

L_p Sampler

1) For $0 < p < 2$, set $\eta = \Theta(1)$, and for $p = 2$ set $\eta = \Theta(\sqrt{1/\log(n)})$. Let $d = \Theta(\log(n))$, and instantiate a $d \times 6/\eta^2$ count-sketch table A, and set $\mu \sim \text{Uniform}[\frac{1}{2}, \frac{3}{2}]$.
2) Duplicate updates to f to obtain the vector $F \in \mathbb{R}^{n^c}$ so that $f_i = F_{i_j}$ for all $i \in [n]$ and $j = 1, 2, \ldots, n^{c-1}$, for some fixed constant c.
3) Choose i.i.d. exponential random variables $t = (t_1, t_2, \ldots, t_{n^c})$, and construct the stream $\zeta_i = F_i \cdot \texttt{rnd}_\nu(1/t_i^{1/p})$.
4) Run A on ζ to obtain estimate y with $|y - |\zeta|\|_\infty < \eta \|\zeta_{\text{tail}(1/\eta^2)}\|_2$ as in Theorem 1.
5) Run L_2 estimator on ζ, and L_p estimator on f, to obtain $R \in [\frac{1}{2}\|\zeta\|_2, 2\|\zeta\|_2]$ and $R' \in [\frac{1}{2}\|F\|_p, 2\|F\|_p]$ with high probability.
6) If $y_{(1)} - y_{(2)} < 100\mu\eta R + \eta R'$ or if $y_{(2)} < 50\eta\mu R$, report FAIL, else return $i \in [n]$ such that $y_{i_j} = y_{(1)}$ for some $j \in [n^{c-1}]$.

Fig. 3: Our main L_p Sampling algorithm

$\sum_{j \in [n^{c-1}]} |F_{i_j}|^p / \|F\|_p^p = |f_i|^p / \|f\|_p^p$, it will therefore suffice to find $i_j \in [n^c]$ such that $i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}$, and return the index $i \in [n]$. The assumption that the t_i's are i.i.d. will later be relaxed in Section V while derandomizing the algorithm.

Now fix any sufficiently large constant c, and fix $\nu > n^{-c}$. To speed up the update time, instead of explicitly scaling F_i by $1/t_i^{1/p}$ to construct the stream z, our algorithm instead scales F_i by $\texttt{rnd}_\nu(1/t_i^{1/p})$, where $\texttt{rnd}_\nu(x)$ rounds $x > 0$ down to the nearest value in $\{\ldots, (1+\nu)^{-1}, 1, (1+\nu), (1+\nu)^2, \ldots\}$. In other words, $\texttt{rnd}_\nu(x)$ rounds x down to the nearest power of $(1+\nu)^j$ (for $j \in \mathbb{Z}$). This results in a separate stream $\zeta \in \mathbb{R}^N$ where $\zeta_i = F_i \cdot \texttt{rnd}_\nu(1/t_i^{1/p})$. Note $\zeta_i = (1 \pm O(\nu))z_i$ for all $i \in [N]$.

Having constructed the transformed stream ζ, we then run a $\Theta(\log(n)) \times 6/\eta^2$ instance A of count-sketch on ζ to obtain an estimate vector y with $\|y - |\zeta|\|_\infty < \eta\|\zeta_{\text{tail}(1/\eta^2)}\|_2$ with probability $1 - n^{-c}$ (as in Theorem 1). Note that y_j is an estimate of the *absolute value* ζ_j, and is always positive. Then for $0 < p < 2$, we set $\eta = \Theta(1)$, and for $p = 2$ we set $\eta = \Theta(1/\sqrt{\log(n)})$. Next, we obtain estimates $R \in [\frac{1}{2}\|\zeta\|_2, 2\|\zeta\|_2]$ via the algorithm of Lemma 1, and $R' \in [\frac{1}{2}\|F\|_p, 2\|F\|_p]$ via Lemma 2, both with high probability. The estimate R' is obtained by estimating $\|f\|_p$ to a constant factor and scaling up by $n^{(c-1)/p}$ (noting that $|F|_p = n^{(c-1)/p}\|f\|_p$). The algorithm then finds $y_{(1)}, y_{(2)}$ (the two largest coordinates of y), and samples $\mu \sim \text{Uniform}[1/2, 3/2]$. It then checks if $y_{(1)} - y_{(2)} < 100\mu\eta R + \eta R'$ or if $y_{(2)} < 50\eta\mu R$, and reports FAIL if either occur, otherwise it returns $i \in [n]$ with $y_{i_j} = y_{(1)}$ for some $j \in [n^{c-1}]$.

Analysis.: Let $i^* \in [n^c]$ be the index of the maximizer in y, so $y_{i^*} = y_{(1)}$. By checking that $y_{(1)} - y_{(2)} > 100\mu\eta R$, noting that $100\mu\eta R \geq 25\|y - |\zeta|\|_\infty$ and $z_k = (1 \pm \nu)\zeta_k$ for all $k \in [n^c]$, for $\nu < \eta$ sufficiently small we ensure that $|z_{i^*}|$ is also the maximum element in z. The necessity for the tests $y_{(2)} \geq 50\eta\mu R$ and $y_{(1)} - y_{(2)} \geq \eta R'$ is technical and less straightforward, and is discussed further in Section V of the full version.

To prove correctness, we then need to analyze the conditional probability of failure given $D(1) = i$. Let $N = |\{i \in [n^c] \mid F_i \neq 0\}|$ (N is the support size of F). We can assume that $N \neq 0$ (to check this one could run, for instance, the $O(\log^2(n))$-bit support sampler of [35]). Note that $n^{c-1} \leq N \leq n^c$. For $1 \leq k \leq n^c$, we define the k-th anti-rank $D(k)$ (as in Section III) of the stream vector z to be the index such that $|z_{D(k)}|$ is the k-th largest value of the $|z_i|$'s. For the following lemma, it suffices to consider only the exponentials t_i with $F_i \neq 0$, and we thus consider only values of k between 1 and N. Thus $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(N)}|$. Moreover, we have that $|z_{D(k)}|^{-p} = \frac{t_{D(k)}}{|F_{D(k)}|^p}$ is the k-th smallest of all the $\frac{t_i}{|F_i|^p}$'s, and by the results of the previous section can be written as $|z_{D(k)}|^{-p} = \sum_{\tau=1}^k \frac{E_\tau}{\sum_{j=\tau}^N |F_{D(j)}|^p}$ where the E_τ are i.i.d. exponentials and independent of the anti-rank vector D.

The following lemma states that for all but the smallest $n^{9c/10}$ (in absolute value) of the z_i's, the random variable $|z_{D(k)}|$ is highly concentrated around a value which is *independent of the anti-rank vector* D. In other words, for such k, $|z_{D(k)}|$ can be written as the sum of a random variable that is independent of the values of $D(i)$ for all $i \in [N]$, and a second adversarial error term which has magnitude at most an $n^{-c/10}$ factor times the first. To prove the lemma, we first state the following straightforward fact.

Fact 5. *For $p \in (0, 2]$, suppose that we choose the constant c such that $mM \leq n^{c/20}$, where note that we have $|F_i| \leq mM$ for all $i \in [N]$. Then if $S \subset \{i \in [n^c] \mid F_i \neq 0\}$ is any subset, then $\sum_{i \in S} |F_i|^p \geq \frac{|S|}{N} n^{-c/10}\|F\|_p^p$*

Proof. We know that $|F_i|^p \leq (mM)^p \leq n^{c/10}$ using $p \leq 2$. Then each non-zero value $|F_i|^p$ is at most an $n^{-c/10}$ fraction of any other item $|F_j|^p$, and in particular of the average item weight. It follows that $|F_i|^p \geq n^{-c/10}\frac{\|F\|_p^p}{N}$ for all $i \in [N]$, which results in the stated fact. □

Lemma 3. *For every $1 \leq k < N - n^{9c/10}$, we have*

$$|z_{D(k)}| = \Big[(1 \pm O(n^{-c/10}))\sum_{\tau=1}^k \frac{E_\tau}{\mathbb{E}[\sum_{j=\tau}^N |F_{D(j)}|^p]}\Big]^{-1/p}$$

with probability $1 - O(e^{-n^{c/3}})$.

Proof. Let $\tau < N - n^{9c/10}$. We can write $\sum_{j=\tau}^N |F_{D(j)}|^p$ as a deterministic function $\psi(t_1, \ldots, t_N)$ of the random scaling

exponentials $t_1, \ldots, t_N$ corresponding to $F_i \neq 0$. We first argue that

$$|\psi(t_1, \ldots, t_N) - \psi(t_1 \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_N)| < 2 \max_j \{F_j^p\}$$
$$< 2n^{-c+1}\|F\|_p^p$$

This can be seen from the fact that changing a value of t_i can only have the effect of adding (or removing) $|F_i|^p$ to the sum $\sum_{j=\tau}^N |F_{D(j)}|^p$ and removing (or adding) a different $|F_l|$ from the sum. The resulting change in the sum is at most $2\max_j\{|F_j|^p\}$, which is at most $2n^{-c+1}\|F\|_p^p$ by duplication. Set $T = N - \tau + 1$. Since the t_i's are independent, we apply McDiarmid's inequality (Fact 1) to obtain $\Pr\Big[\big|\sum_{j=\tau}^N |F_{D(j)}|^p - \mathbb{E}[\sum_{j=\tau}^N |F_{D(j)}|^p]\big| > \epsilon T n^{-c}\|F\|_p^p\Big] \leq 2\exp\big(\frac{-2\epsilon^2 T^2 n^{-2c}\|F\|_p^{2p}]}{n^c(2n^{-c+1}\|F\|_p^p)^2}\big) \leq 2\exp\big(-\frac{1}{2}\epsilon^2 T^2 n^{-c-2}\big)$. Setting $\epsilon = \Theta(n^{-c/5})$ and using $T > n^{9c/10}$, this is at most $2\exp(-\frac{1}{2}n^{2c/5-2})$.

To show concentration up to a $(1 \pm O(n^{-c/10}))$ factor, it remains to show that $\mathbb{E}[\sum_{j=\tau}^N |F_{D(j)}|^p] = \Omega(Tn^{-11c/10}\|F\|_p^p)$. This follows from the Fact 5, which gives $\sum_{j=0}^T |F_{D(-j)}|^p \geq n^{-c/10}(Tn^{-c}\|F\|_p^p)$ deterministically. Now recall that $|z_{D(k)}| = [\sum_{\tau=1}^k \frac{E_\tau}{\sum_{j=\tau}^N |F_{D(-j)}|^p}]^{-1/p}$. We have just shown that $\sum_{j=\tau}^N |F_{D(j)}|^p = (1 \pm O(n^{-c/10}))\mathbb{E}[\sum_{j=\tau}^N |F_{D(j)}|^p]$, so we can union bound over all $\tau = 1, 2, \ldots, N - n^{9c/10}$ to obtain

$$|z_{D(k)}| = \Big[(1 \pm O(n^{-c/10}))\sum_{\tau=1}^k \frac{E_\tau}{\mathbb{E}[\sum_{j=\tau}^N |F_{D(j)}|^p]}\Big]^{-1/p}$$

for all $k \leq N - n^{9c/10}$ with probability $1 - O(n^c e^{-n^{2c/5-2}}) = 1 - O(e^{n^{c/3}})$. □

We use this result to show that our failure condition is nearly-independent of the value $D(1)$, as stated formally in the following Lemma. Let $\neg$FAIL be the event that the algorithm L_p Sampler does not output FAIL. The following is the main technical result of the paper.

Lemma 4. *For $p \in (0, 2]$ a constant bounded away from 0 and any $\nu \geq n^{-c}$,*

$$\Pr[\neg\texttt{FAIL} \mid D(1)] = \Pr[\neg\texttt{FAIL}] \pm O(\log(n)\nu)$$

for every possible value of $D(1) \in [N]$.

Lemma 4 demonstrates that the probability of failure can only change by an additive $O(\log(n)\nu)$ term given that any particular value of $i \in [N]$ achieved the maximum (i.e. $D(1) = i$). This property will translate into a $(1 \pm \log(n)\nu)$-*relative error* in our sampler, where the space complexity is independent of ν. To complete the proof of correctness of our algorithm, we now need to bound the probability that we fail at all. The following Lemma does precisely this.

Lemma 5. *If y is the vector obtained via count-sketch as in the algorithm L_p Sampler, and $0 < p \leq 2$ a constant, then we have $\Pr[y_{(1)} - y_{(2)} > 100\eta\mu R + \eta R',\ y_{(2)} > 50\eta\mu R] \geq 1/2$, where $\eta = \Theta(1)$ when $p < 2$, and $\eta = \Theta(1/\sqrt{\log(n)})$ when $p = 2$.*

Putting together the results of this section, we obtain the correctness of our algorithm as stated in Theorem 2. In Section V, we will show that the algorithm can be implemented to have $\tilde{O}(1)$ update and query time, and that the entire algorithm can be derandomized to use $O(\log^2(n))$ bits of space for $p \in (0, 2)$ and $O(\log^3(n))$ bits for $p = 2$.

Theorem 2. *Given any constant $c \geq 2$, $\nu \geq n^{-c}$, and $0 < p \leq 2$, there is a one-pass L_p sampler which returns an index $i \in [n]$ such that $\Pr[i = j] = \frac{|f_j|^p}{\|F\|_p^p}(1 \pm \nu)$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta))$ bits for $p < 2$, and $O(\log^3(n)\log(1/\delta))$ bits for $p = 2$. The update time is $\tilde{O}(\nu^{-1})$, and the query time is $\tilde{O}(1)$.*

In particular, it follows that perfect L_p samplers exist using $O(\log^2(n)\log(1/\delta))$ and $O(\log^3(n)\log(1/\delta))$ bits of space for $p < 2$ and $p = 2$ respectively.

Theorem 3. *Given $0 < p \leq 2$, for any constant $c \geq 2$ there is a perfect L_p sampler which returns an index $i \in [n]$ such that $\Pr[i = j] = \frac{|f_j|^p}{\|F\|_p^p} \pm O(n^{-c})$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta))$ bits for $p < 2$, and $O(\log^3(n)\log(1/\delta))$ bits for $p = 2$.*

Remark 1. Note that for p arbitrarily close to 2, the bound given by Lemma 5 degrades. In this case, we set $\eta = \Theta((\log(n))^{-1/2})$, and use the upper bound for $p = 2$. Similarly, for p arbitrarily close to 0, the bound also degrades. For such non-constant p arbitrarily close to 0, we direct the reader to the $O(\log^2(n))$-bit perfect L_0 sampler of [35].

V. Time and Space Complexity

The purpose of this section is to show how our main sampling algorithm can be implemented with the runtime as stated in Theorem 2 without increasing the space complexity. Recall that our algorithm utilizes two data structures on the stream ζ: the $d \times 6/\eta^2$ count-sketch matrix A, and the vector $B \in \mathbb{R}^r$ (with $r = \Theta(\log(n))$) as in Lemma 1 which is used to produce the estimate R of $\|\zeta\|_2$. We note that the update and query time of the L_p estimation algorithm which obtains $R' \in [\frac{1}{2}\|F\|_p, 2\|F\|_p]$ is already $O(1)$ by Lemma 1, since we can simply estimate the value $\|f\|_p$ to a constant factor and scale by $n^{(c-1)/p}$, since we have $n^{(c-1)/p}\|f\|_p = \|F\|_p$ exactly. So we need not consider the update or query time of this L_p estimation algorithm.

Our results can be broken into three parts. First, we introduce an update procedure Fast-Update to update both A, B throughout the stream, and show that implementing L_p Sampler with Fast-Update results in $\tilde{O}(\nu^{-1})$ update time. This result is as follows.

Lemma 6. *There is a procedure, Fast-Update, such that running the L_p sampler with the update procedure given by Fast-Update results in the same distribution over the*

count-sketch table A and L_2 estimation vector B as the original algorithm. Moreover, conditioned on a fixed realization of A, B, the output of the original algorithm will be the same as the output of the algorithm using `Fast-Update`*. For a given $i \in [n]$,* `Fast-Update` *requires $\tilde{O}(\nu^{-1})$ random bits, and runs in time $\tilde{O}(\nu^{-1})$.*

This result gives the stated update time of Theorem 2, but requires too many random bits to be stored. Thus we next show that the algorithm L_p `Sampler` with `Fast-Update` can be derandomized to use a random seed of length $O(\log^2(n))$-bits, which will give the desired space complexity. The result follows from a careful analysis of the exact seed length needed by Nisan's pseudorandom generator to fool a statistical tester of our algorithm.

Lemma 7. *The algorithm L_p* `Sampler` *using* `Fast-Update` *as its update procedure can be derandomized using a random seed of length $O(\log^2(n))$. Moreover, this does not effect the time complexity as stated in Lemma 6*

Finally, we show how using the additional auxiliary heavy-hitters data structure of [39] (called `ExSk` here) as a subroutine, we can obtain the stated $\tilde{O}(1)$ update time as well. This additional data structure will not increase the space or update time complexity of the entire algorithm, and does not need to be derandomized.

Lemma 8. *For any constant $c > 0$, with probability $1 - n^{-100c}$ the algorithm L_p* `Sampler` *with* `Fast-Update` *and* `ExSk` *returns the same output (an index $i \in [n]$ or* `FAIL`*) as L_p* `Sampler` *using* `Fast-Update` *but without* `ExSk`*. The space and update time are not increased by using* `ExSk`*, and the query times is now $\tilde{O}(1)$.*

VI. Estimating the Frequency of Outputted Coordinate

We now describe our results for frequency estimation of the sampled coordinate f_i. Specifically, conditioned on our algorithm L_p `Sampler` returning an index $i \in [n]$, we can obtain an estimate $\tilde{f}_i = (1 \pm \epsilon) f_i$ with probability $1 - \delta_2$. We now describe how to do this algorithmically, leaving the analysis to the full version. Given a parameter $\epsilon > 0$, our algorithm, in addition to the count-sketch matrix A used by L_p `Sampler`, stores a second count-sketch matrix A' with, $d' = O(\log(1/\delta_2))$ rows and $\gamma = O(\min\{\epsilon^{-2}, \max\{\epsilon^{-p}, \log(\frac{1}{\delta_2})\}\}\eta^{-2})$ columns, where η is the parameter used in L_p `Sampler`. We then update A' in the same way we update A via the `Fast-Update` routine given in Section V.

When L_p `Sampler` returns some $i_j \in [n^c]$ (corresponding to some duplicate i_j of i), then our estimate of f_i is $\tilde{f} = \tilde{y}_{i_j}(\mathtt{rnd}_\nu(1/t_{i_j}))^{-1}$, which we will show satisfies $\tilde{f} = (1 \pm \epsilon) f_i$. Putting this together with Theorem 2, we will obtain the following result:

Theorem 4. *There is an algorithm $\mathcal{A}$ which, on a general turnstile stream f, outputs $i \in [n]$ with probability $|f_i|^p/\|f\|_p^p(1 \pm \nu) + O(n^{-c})$, and outputs* `FAIL` *with probability at most δ_1. Conditioned on outputting some $i \in [n]$, $\mathcal{A}$ will then output $\tilde{f}$ such that $\tilde{f} = (1 \pm \epsilon) f_i$ with probability $1 - \delta_2$. The space required is $O\big(\big(\log^2(n) + \beta \log(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p \in (0, 2)$, and $O\big(\big(\log^3(n) + \epsilon^{-2}\log^2(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p = 2$, where $\beta = \min\{\epsilon^{-2}, \max\{\epsilon^{-p}, \log(\frac{1}{\delta_2})\}\}$. The update time is $\tilde{O}(\nu^{-1})$ and the query time is $\tilde{O}(1)$.*

VII. Lower bounds

We now state our lower bound for providing relative error approximations of the frequency of a sampled item. In fact, we are able to show that even when the index output is from a distribution with constant *additive* error from the true L_p distribution, returning an estimate with probability $1 - \delta_2$ still requires $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space.

Theorem 5. *Fix any $p > 0$ constant bounded away from 0, and let $\epsilon < 1/3$ with $\epsilon^{-p} = o(n)$. Then any L_p sampling algorithm that outputs FAIL with probability at most $1/100$, and otherwise returns an item $\ell \in [n]$ such that $\Pr[\ell = l] = |f_l|^p/\|f\|_p^p \pm 1/50$ for all $l \in [n]$, along with an estimate $\tilde{f}_\ell$ such that $\tilde{f}_\ell = (1 \pm \epsilon) f_\ell$ with probability $1 - \delta_2$, requires $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space.*

VIII. Conclusion

This work demonstrates the existence of perfect L_p samplers for $p \in (0, 2)$ using $O(\log^2(n)\log(1/\delta))$ bits of space. This bound is tight in terms of both n and δ. However, there are several open problems for L_p samplers which this work does not close. Notably, there is still a $\log(n)$ gap between the upper and lower bounds for L_2 samplers, as the best known lower bound for any $p \geq 0$ is $\Omega(\log^2 n)$, while our upper bound is $O(\log^3 n)$. While perfect L_2 samplers using polylogarithmic space were not known before this work, our upper bound matches the best upper bounds of prior approximate L_2 samplers with constant $\nu = \Omega(1)$. It is therefore an open question whether this additional factor of $\log n$ is required in the space complexity of an L_2 sampler, perfect or otherwise.

Secondly, one notable shortcoming of the perfect sampler presented in this paper is the very large update time. To obtain a perfect sampler as defined in the introduction, the algorithm in this paper takes polynomial (in n) time to update its data structures after each entry in the stream. This is clearly non-ideal, since most streaming applications demand constant or polylogarithmic update time. Using our rounding procedure, we can obtain a $(1 \pm 1/\mathrm{poly}(\log n))$ relative error sampler with polylogarithmic update time (and the same space as the perfect sampler), but it is still an open problem to design a perfect L_p sampler with optimal space dependency as well as polylogarithmic update time.

Finally, there are several gaps in the dependency on ϵ, δ_2 in our procedure which, in addition to outputting a index $i \in [n]$, also outputs a $(1 \pm \epsilon)$ estimate of the frequency f_i. Taking Theorem 5 along with the known lower bounds for L_p sampling, our best lower bound for the problem is $\Omega(\log^2(n)\log(1/\delta_1) + \epsilon^{-p}\log(n)\log(1/\delta_2))$,

where δ_1 is the probability that the sampler fails to output an index i. On the other hand, our best upper bound is $O\big(\big(\log^2(n) + \beta \log(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p \in (0,2)$, and $O\big(\big(\log^3(n) + \epsilon^{-2}\log^2(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p = 2$, where $\beta = \min\big\{\epsilon^{-2}, \max\{\epsilon^{-p}, \log\big(\frac{1}{\delta_2}\big)\}\big\}$. Notably, the $\log(1/\delta_1)$ multiplies the $\log(1/\delta_2)$ term in the upper bound but not in the lower bound, and when $\log(1/\delta_2) > \epsilon^{-p}$ the ϵ dependency is not tight. We leave it as an open problem to determine precisely the right dependencies of such an algorithm on $\epsilon, \delta_1, \delta_2$.

REFERENCES

[1] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. *arXiv preprint arXiv:1011.1263*, 2010.

[2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[3] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–634. Society for Industrial and Applied Mathematics, 2002.

[4] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 740–753. ACM, 2016.

[5] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.

[6] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, London, UK, UK, 2002. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646255.684566.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, pages 784–784, 2002.

[8] 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. ACM, 2015.

[9] Edith Cohen, Graham Cormode, and Nick Duffield. Don't let the negatives bring you down: sampling from streams of signed updates. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):343–354, 2012.

[10] Edith Cohen, Graham Cormode, and Nick G. Duffield. Structure-aware sampling: Flexible and accurate summarization. *PVLDB*, 4(11):819–830, 2011. URL: http://www.vldb.org/pvldb/vol4/p819-cohen.pdf.

[11] Edith Cohen, Nick Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Stream sampling for variance-optimal estimation of subset sums. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1255–1264. Society for Industrial and Applied Mathematics, 2009.

[12] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Algorithms and estimators for summarization of unaggregated data streams. *J. Comput. Syst. Sci.*, 80(7):1214–1244, 2014. URL: https://doi.org/10.1016/j.jcss.2014.04.009, doi:10.1016/j.jcss.2014.04.009.

[13] Graham Cormode, S Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st international conference on Very large data bases*, pages 25–36. VLDB Endowment, 2005.

[14] Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 77–86. ACM, 2010.

[15] Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Continuous sampling from distributed streams. *Journal of the ACM (JACM)*, 59(2):10, 2012.

[16] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[17] Nick Duffield. Sampling for passive internet measurement: A review. *Statistical Science*, pages 472–498, 2004.

[18] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003. URL: http://doi.acm.org/10.1145/859716.859719, doi:10.1145/859716.859719.

[19] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

[20] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[21] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 595–606, 2006. URL: http://dl.acm.org/citation.cfm?id=1164179.

[22] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *VLDB J.*, 17(2):173–202, 2008. URL: https://doi.org/10.1007/s00778-007-0065-y, doi:10.1007/s00778-007-0065-y.

[23] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 331–342, 1998. URL: http://doi.acm.org/10.1145/276304.276334, doi:10.1145/276304.276334.

[24] Phillip B Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *ACM SIGMOD Record*, volume 27, pages 331–342. ACM, 1998.

[25] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms.

[26] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin Strauss. Quicksand: Quick summary and analysis of network data. Technical report, 2001.

[27] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 454–465. Elsevier, 2002.

[28] Peter J. Haas. Data-stream sampling: Basic techniques and results. In *Data Stream Management - Processing High-Speed Data Streams*, pages 13–44. 2016. URL: https://doi.org/10.1007/978-3-540-28608-0_2, doi:10.1007/978-3-540-28608-0_2.

[29] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Arun N Swami. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550–569, 1996.

[30] Peter J Haas and Arun N Swami. *Sequential sampling procedures for query size estimation*, volume 21. ACM, 1992.

[31] Ling Huang, XuanLong Nguyen, Minos Garofalakis, Joseph M Hellerstein, Michael I Jordan, Anthony D Joseph, and Nina Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 134–142. IEEE, 2007.

[32] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[33] Rajesh Jayaram and David P Woodruff. Perfect lp sampling in a data stream. *arXiv preprint*, 2018.

[34] Thathachar S Jayram and David P Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):26, 2013.

[35] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/1989284.1989289, doi:10.1145/1989284.1989289.

[36] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Pro-*

ceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms, pages 1161–1178. SIAM, 2010.

[37] Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. *arXiv preprint arXiv:1704.00633*, 2017.

[38] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. URL: http://www.worldcat.org/oclc/312898417.

[39] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 61–70. IEEE, 2016.

[40] Richard J Lipton and Jeffrey F Naughton. Query size estimation by adaptive sampling. *Journal of Computer and System Sciences*, 51(1):18–25, 1995.

[41] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. *Practical selectivity estimation through adaptive sampling*, volume 19. ACM, 1990.

[42] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM, 2006.

[43] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. *PVLDB*, 5(12):1699, 2012. URL: http://vldb.org/pvldb/vol5/p1699_gurmeetsinghmanku_vldb2012.pdf.

[44] Colin McDiarmid. *On the method of bounded differences*, page 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press, 1989. doi:10.1017/CBO9781107359949.008.

[45] Morteza Monemizadeh and David P Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010.

[46] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[47] HN Nagaraja. Order statistics from independent exponential random variables and the sum of the top order statistics. *Advances in Distribution Theory, Order Statistics, and Inference*, pages 173–185, 2006.

[48] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[49] Frank Olken. *Random sampling from databases*. PhD thesis, University of California, Berkeley, 1993.

[50] Marina Thottan, Guanglei Liu, and Chuanyi Ji. Anomaly detection approaches for communication networks. In *Algorithms for Next Generation Networks*, pages 239–261. Springer, 2010.

[51] Srikanta Tirthapura and David P Woodruff. Optimal random sampling from distributed streams revisited. In *International Symposium on Distributed Computing*, pages 283–297. Springer, 2011.

[52] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[53] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. URL: http://doi.acm.org/10.1145/3147.3165, doi:10.1145/3147.3165.

[54] David P. Woodruff and Qin Zhang. Subspace embeddings and l_p-regression using exponential random variables. *CoRR*, abs/1305.5580, 2013. URL: http://arxiv.org/abs/1305.5580, arXiv:1305.5580.

[55] David P Woodruff and Peilin Zhong. Distributed low rank approximation of implicit functions of a matrix. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 847–858. IEEE, 2016.