L_p Samplers and Their Applications: A Survey

GRAHAM CORMODE, University of Warwick
HOSSEIN JOWHARI, K. N. Toosi University of Technology

The notion of L_p sampling, and corresponding algorithms known as L_p samplers, have found a wide range of applications in the design of data stream algorithms and beyond. In this survey we present some of the core algorithms to achieve this sampling distribution based on ideas from hashing, sampling and sketching. We give results for the special cases of insertion-only inputs, lower bounds for the sampling problems, and ways to efficiently sample multiple elements. We describe a range of applications of L_p drawing on problems across the domain of computer science, from matrix and graph computations, as well as to geometric and vector streaming problems.

CCS Concepts: • **Theory of computation → Sketching and sampling**;

ACM Reference format:
Graham Cormode and Hossein Jowhari. 2018. L_p Samplers and Their Applications: A Survey. 1, 1, Article 1 (November 2018), 35 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Random sampling is an indispensable tool in computation over massive data sets. In addition to its wide applications in generating small summaries of data, it also acts as the main building block in the design of many algorithms and estimation procedures. However despite its simplicity and common use, sampling turns into a challenging puzzle when it is applied to distributed and dynamic data sets. Specifically in the area of data streams and distributed computing where serious limitations on storage space and communication cost are imposed, a naive implementation of sampling could take a lot of resources and even in some cases become impractical. For instance, consider the scenario where an Internet Service Provider with tens of millions of clients requires samples from the recent activities of its clients for performance tuning purposes. Or consider the case where a financial institution demands samples from the recent updates (withdrawals and deposits) on the current accounts of its clients. In both cases data is constantly evolving and possibly distributed across several servers. Sampling methods that assume data is static and resides in the main memory are not applicable

This work is supported in part by European Research Council grant ERC-2014-CoG 647557, The Alan Turing Institute under EPSRC grant EP/N510129/1 and a Royal Society Wolfson Research Merit Award.

XXXX-XXXX/2018/11-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

in these scenarios. For another example consider a data stream that corresponds to all visits to a website. We would like every visitor to have some chance to be sampled, but for those who visit more frequently to be disproportionately more likely to be selected—say, proportional to the square of the number of visits. Sampling with such a distribution under space limitations can be difficult to achieve if the visits are spread out in time, and the actual number of visits accumulates gradually.

To tackle these challenges, the model of L_p *sampling* has emerged as a way to capture a variety of sampling distributions and scenarios, including those described above. Concurrently, a number of algorithms and applications that (approximately) achieve the required sampling distributions are known. Informally, for a parameter p, an L_p sampler should sample an element proportional to the p'th power of its frequency in a stream of observations. Note that the stream of observations may include deletions of already observed data which is in particular hard to handle under strict memory limitations. To address this, techniques and data summaries are used that support linear operations—i.e., given two independent streams of updates, we can add/substract the corresponding (summaries) sketches to get a sample from the union/difference of the streams. In fact it is this linear property of such samplers that makes them very powerful in practice.

To explain the model more formally we start with a key definition.

Definition 1.1. Let $x \in \mathbb{R}^n$ be a non-zero vector. For $p > 0$, the (exact) L_p *distribution* corresponding to x is a distribution on $[n]$ that takes i with probability $\frac{|x_i|^p}{\|x\|_p^p}$, where $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$ is the L_p norm of x. For the limiting case of $p = 0$, we define the L_p distribution corresponding to x to be the uniform distribution over non-zero coordinates of x.

A *perfect L_p-sampler* is a randomized algorithm such that given a stream of updates on the coordinates of an initially zero vector x, it outputs a pair (i, x_i) where i is distributed according to the L_p distribution of x and fails only if x is the zero vector. Note that for the stream of observations the updates correspond to the additions and subtractions of the individual element frequencies.

While we do not know how to achieve exactly this distribution without having the entire vector x kept in memory, fortunately there are polylogarithmic space algorithms that get arbitrary close. It is worth mentioning that all of the applications of these samplers that we know of do not require the exact distribution. In fact it would be interesting to identify a (non-artificial) application where this requirement is strongly required.

Thus an *approximate L_p-sampler* is allowed some probability of failure, and the distribution of its output is required to be close to the L_p distribution. More formally, an approximate L_p sampler is parameterized as follows.

Definition 1.2. An approximate L_p sampler with relative error ε and additive error Δ, conditioned on no failure, outputs the index $i \in [n]$ with a probability p_i in the range

$$p_i \in (1 \pm \varepsilon)\frac{|x_i|^p}{\|x\|_p^p} \pm \Delta,$$

i.e. we tolerate some imprecision in the sampling probability compared to the target distribution. The sampler is often implemented as a randomized algorithm, and so can have an additional paramter δ, meaning that it may fail with probability at most δ and output no sample at all*

We remark that in almost all existing samplers the additive error parameter Δ can be reduced to $O(n^{-c})$ where c is an arbitrary constant without affecting the space and update time complexities. Therefore in the rest of the paper when we refer to an approximate L_p sampler algorithm, it is implicitly the case that $\Delta = O(n^{-1})$ unless it is explicitly stated otherwise.

Throughout the article, we maintain the distinction that L_p sampling refers to the objective of sampling according to the (approximate) L_p distribution, while L_p samplers are algorithms (and their corresponding data structures) which allow this sampling to be performed. Therefore, we study L_p samplers through the lens of computational complexity: what is the space required (as a function of n, ε and δ) by the algorithm; what is the time required to process an update in the data stream; and what is the time required to extract a sample from the stored data structure. We focus on the first two of these criteria, and look for L_p samplers with fast update time and small space usage.

The current state of the art for approximate L_p samplers is shown in Figure 1. Here, we distinguish between the general update case (updates on x_i's can be both positive and negative), and the positive update case (negative updates are disallowed). Each of these results is described in more detail throughout this survey. The table uses a parameter m to help draw out the similarities between the various costs: for each method, m gives the leading term for the space and time cost, and suppresses lower order factors in $\log n$ and $\log 1/\delta$). The fact that we can write the costs in this form across different values of m is primarily due to similarities in the underlying algorithms and their constructions – there does not seem to be any greater significance of m than notational convenience.

1.1 Preliminaries and Related Topics

The treatment in this survey assumes some level of familiarity with concepts from randomized algorithms, such as hash functions to map data elements in a pseudo-random but repeatable fashion, and tools for analysis from concentration of measure. We present definitions and basic results below, and refer to textbooks such as those of Motwani and Raghavan [64] and Mitzenmacher and Upfal [62] for further details.

Data model. Throughout this article, we consider data that can be modeled as a vector x of frequencies. The vector x is defined by a sequence of (integral) increments or decrements to individual coordinates. That is, $x \in \mathbb{Z}_M^n$ where $\mathbb{Z}_M$ is the set of integers within the range $[-M, M]$ where $M = n^{O(1)}$. We also restrict the updates to the coordinates of x to integers within the same range. These assumptions capture most of the applications of L_p sampling, while allowing a broad range of algorithms.

Limited independence hash functions. Algorithms for L_p sampling are inherently randomized, and are often defined using hash functions h, which map input values

*Note that we could consider this equivalent to setting $\delta = \Delta$ and augmenting the output space with an element $\emptyset$ such that $x_\emptyset = 0$. However, for clarity, we keep these parameters separate.

p	$(\varepsilon, \Delta, \delta)$	space usage	update time	Note
General Updates		$O(m \log^2 n \log 1/\delta)$	$O(m \log n \log 1/\delta)$	
$p = 0$	$(0, n^{-2}, \delta)$	$m = 1$		[49],[23]
$p \in (0,2)/\{1\}$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{1}{\varepsilon^p}$		[49]
$p = 1$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{\log(1/\varepsilon)}{\varepsilon}$		[49]
$p = 2$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{1}{\varepsilon^2} \log n$		[49],[7]
$p \in (0,2)$	$(0, 0, \delta)$	$m = 1$		[46]
$p = 2$	$(0, 0, \delta)$	$m = \log n$		[46]
Positive Updates		$O(m \log m \log n \log 1/\delta)$	$O(m \log m \log 1/\delta)$	
$p \in (0,2)/\{1\}$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{1}{\varepsilon^p}$		[49],[11]
$p = 1$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{\log(1/\varepsilon)}{\varepsilon}$		[49],[11]
$p = 2$	$(\varepsilon, n^{-2}, \delta)$	$m = \frac{1}{\varepsilon^2} \log n$		[7],[49],[11]
Special Cases				
$p = 0$	$(\varepsilon, 0, 0)$	$O\left(\frac{\log(1/\varepsilon)}{\varepsilon} \log n\right)$	$O\left(\frac{\log(1/\varepsilon)}{\varepsilon}\right)$	Min-wise sampling
$p = 1$	$(0, 0, 0)$	$O(\log n)$	$O(1)$	Reservoir sampling
$p = 2$, unit weights	$(0, 0, \delta)$	$O(\sqrt{n} \log 1/\delta)$	$O(1)$	Random sampling

Fig. 1. State of the art for approximate L_p samplers.

from a domain of size n to a bounded output domain of size d. It would be convenient to assume that hash functions are drawn uniformly from the space of all functions. However, we are often concerned with the space used by our algorithms. To describe a random function requires a large amount of space, since we need to encode the output value for each distinct input. Consequently, we will often rely on *limited independence* hash functions, drawn from a smaller family. These can be described more compactly, but are sufficiently random to allow formal guarantees to be proven. A common requirement is for k-wise independence: over the random choice of the hash function, the probability that any k distinct elements are mapped to a particular combination of output values is uniform, i.e. $1/d^k$. Such hash functions are easy to construct.

Lemma 1.3. *[16] A function drawn from a family of k-wise independent hash functions can be encoded in $O(k \log n)$ bits.*

The subsequent lemma is used in the analysis of some of the constructions.

Lemma 1.4. *[69, Thm. 5] If X is the sum of k-wise random variables each of which is confined to the interval $[0,1]$ with $\mu = \mathsf{E}[X]$, then for $\delta \geq 0$ and $k \leq \lfloor \min\{\delta, \delta^2\} \mu e^{-1/3} \rfloor$, we have $\Pr(|X - \mu| \geq \delta\mu) \leq e^{-\lfloor k/2 \rfloor}$.*

Min-wise Independence. A different model of limited independence hash functions is given by the notion of min-wise independent hash functions, introduced by Broder [13, 14]. These are heavily used to define L_p sampling procedures (and in particular L_0 sampling): algorithms apply a min-wise hash function to permute the input, and keep information about elements which map to the smallest few observed values.

A family of functions $\mathcal{H} = \{h_i : [n] \to [n]\}$ is ε-min-wise independent if for any $X \subset [n]$ and $x \in [n] - X$ we have

$$\Pr_{h \in \mathcal{H}}[h(x) < \min h(X)] = \frac{1}{|X| + 1}(1 \pm \varepsilon).$$

Note that here, the probability is over the random choice of a member h from the family $\mathcal{H}$. When X is restricted to subsets of size at most s, the family is called (ε, s)-min-wise independent. Small (approximate) min-wise independent families of hash functions in particular are useful in design of L_0 samplers. The algorithm samples a hash function from the family and then keeps track of information pertaining to the element from the stream that achieves the smallest hash value. The algorithm only needs to keep a sample hash function and a minimum candidate. As result the space of the algorithm principally depends on the size of the family of the hash function, which determines the space needed to represent the sampled function. We shall see the application of this idea in Section 2.2. The following result by Indyk [45] shows that $O(\log(1/\varepsilon))$-wise independent hash functions are indeed also ε-min-wise hash functions. Given efficient constructions for k-wise independent hash functions, this implies that only $O(\log(1/\varepsilon)\log n)$ bits are needed to store a sample ε-min-wise hash function.

Lemma 1.5. *There exist constants c and $c' > 1$ such that for any $\varepsilon > 0$ and $s \leq \varepsilon n/c$ any $c'\log(1/\varepsilon)$-wise independent family $\mathcal{H}$ of functions is (ε, s)-min-wise independent.*

L_p Approximation and Frequency Moments. Algorithms for estimating the L_p norms of vectors have had a central role in shaping the progress of data stream algorithms. One of the first problems to be addressed in this area is to estimate the k'th frequency moment of an input stream (denoted F_k). The problem is to compute the sum over all characters a of the k'th power of the frequency of a, i.e. $F_k = \sum_a f(a)^k$. If we define x as the frequency vector of the stream, then $F_k = \|x\|_k^k$, the k'th power of its L_k norm. Hence, F_k estimation is often addressed with solutions for L_p estimation.

By definition, L_p sampling has a tight relationship with L_p norms and related algorithms. In fact, L_p sampling can be regarded as an influential byproduct of research in L_p estimation algorithms. One direction is almost immediate. L_p samplers can be used to estimate L_p norms, although in an inefficient way[†]. Not surprisingly, L_p estimators are also used in the design of L_p samplers as a subroutine. In particular the following result is a main ingredient of the algorithm in Section 2. We remark that the factor 2 in the lemma is arbitrary and can be replaced by any constant factor.

[†]For example, we could add an extra coordinate to the input vector with weight w, and repeatedly take samples from this modified vector. The fraction of times that the extra coordinate is returned can be used to infer the total L_p weight of the input vector. One can carry out this process with geometrically increasing parallel guesses for w to ensure that there is some guess of w that is not too large or too small. We later give a more efficient construction to estimate L_p norms via L_2 sampling.

Lemma 1.6. [5, 50] *For any $p \in [0, 2]$, there is a streaming algorithm that processes the vector x defined as a sequence of updates to coordinates. The algorithm stores $O(\log n)$ bits of space and outputs a value r satisfying $\|x\|_p \leq r \leq 2\|x\|_p$ with high probability (i.e. the probability that the estimate fails to fall in this range is polynomially small in n).*

Vector Approximation and Sparse Recovery. L_p sampling also borrows ideas and techniques from vector approximation and sparse recovery. This in turn is another important line of research in computation over data streams and massive data sets. In vector approximation the goal is to recover a vector from a stream of updates using small space with as little error as possible. Well-known measures of error are L_1 and L_2 norms which are implemented by two popular summary structures known as Count-Min [24] and Count-Sketch [17]. In sparse recovery, the aim is to recover a vector that is particularly sparse, meaning that it has a small number of non-zero coordinates. More precisely, for $0 \leq c \leq n$, we call the vector $x \in \mathbb{R}^n$ *c-sparse* if all but at most c coordinates of x are zero. When dealing with vectors that are not strictly c-sparse, we will use a measure of sparsity based on the residual norm after removing the c largest entries. Formally, we define the tail error $\mathrm{Err}_p^c(x) = \min \|x - \hat{x}\|_p$, where $\hat{x} \in \mathbb{R}^n$ ranges over all the c-sparse vectors. As we see in Section 2, some implementations of L_p sampling use Count-Sketch to approximately recover a vector which corresponds to the input with entries rescaled by random factors.

Briefly speaking, the Count-Sketch data structure applied to input vector x with parameter c (denoted by Count-Sketch$_c(x)$) uses $O(\log n)$ random functions to distribute the coordinates among $6c$ different sums. Additionally each coordinate is multiplied with a random number from $\{-1, +1\}$ before it is added to its associated sums. All random hash functions and random coefficients are selected from pairwise independent families. The estimated value of a coordinate is the median value of the corresponding sums (each multiplied by the coordinate's random sign). We have the following lemma regarding the accuracy of Count-Sketch.

Lemma 1.7. [17] *For any $x \in \mathbb{R}^n$ and $c \geq 1$ we have $|x_i - x_i^*| \leq \mathrm{Err}_2^c(x)/c^{1/2}$ for all $i \in [n]$ with high probability, where x^* is the output of the Count-Sketch$_c(x)$. As a consequence we also have*

$$\mathrm{Err}_2^c(x) \leq \|x - \hat{x}\|_2 \leq 10\,\mathrm{Err}_2^c(x)$$

with high probability, where $\hat{x}$ is the c-sparse vector best approximating x^* (i.e., $\hat{x}_i = x_i^*$ for the c coordinates i with $|x_i^*|$ highest and is $\hat{x}_i = 0$ for the remaining $n - c$ coordinates).

1.2 A Brief History

Inspired by its wide range of applications in the design of streaming algorithms, the notion of L_p sampling and L_p samplers were introduced and formalized by Monemizadeh and Woodruff [63]. The authors also showed examples where these samplers are used as building blocks in solutions for some data stream problems. However instances of L_p sampling and in particular L_0 sampling appeared earlier in the context of *distinct sampling* where the goal is to sample each element that appears in the stream of data uniformly, irrespective of the number of times that it appears. Specifically the works of Cormode *et al.* [25] and Frahling *et al.* [37] have tackled L_0 sampling within different contexts and have given similar solutions for this problem. Inspired by database applications, the work of Cormode *et al.* [25] considered querying *the inverse distribution*

of a dynamic frequency vector subject to insertion and deletions of characters. More precisely, if $f(a)$ is the number of occurrences of character a in the stream, we want to to learn about the cardinality of $f^{-1}(i)$, the fraction of characters that appear exactly i times in the stream. For notational convenience, we write $f^{-1}(i)$ to denote this fraction, following [25]. One way to estimate $f^{-1}(i)$, is to get a sample character (along with its count) where each character with non-zero count has equal chance to be sampled. In other words, sampling a random pair $(a, f(a))$ such that $f(a) \neq 0$ with probability $1/k$ where k is the number of non-zero coordinates in f. Similarly the work of Frahling *et al.* [37], inspired by geometric applications, is concerned with getting a sample of points (on a plane for instance) subject to insertion and deletion operations. Here the data set can be viewed as a two-dimensional matrix where each insertion or deletion of a point increments or decrements the value of a corresponding cell and the the goal is to sample a non-zero cell in the matrix. More details of these applications is given below. To date, there has been limited empirical study of L_p sampling, and the impact has primarily been within the context of Theoretical Computer Science. The works of Cormode *et al.* [25] and Cormode and Firmani [23] have provided implementation and evaluation for L_0 sampling, showing that samples can be drawn from this distribution, at a storage cost in the region of tens of kilobytes per sampler.

1.3 The Broader Landscape of Stream Sampling Algorithms

The model of L_p sampling is but one sampling distribution, and many other approaches to sampling have been presented in the computational and statistical literature. Restricting our focus to methods proposed in the context of sampling from streams of data, in this section we provide a brief overview of the landscape, and draw connections to L_p sampling, which remains our main focus.

Reservoir sampling. The simplest sampling objective in streams of data is to maintain a uniform sample of the elements so far (without replacement), while new elements arrive online. This can be achieved quite straightforwardly by an algorithm which maintains a sample with the correct distribution over the stream seen so far, and which then selects each new element into the sample (and ejects a currently sampled element) with the appropriate probability [35, 56, 74]. Extensions to the case where elements arrive with an associated weight is a little more involved, and requires sampling based on random values drawn from the exponential distribution parametrized by the weight of each element [32]. This model of uniform sampling coincides with L_1 sampling in the case of insert-only streams, and leads to a space efficient algorithm, discussed in more detail in Section 2.3.

Other sampling distributions. Across the various areas that have considered the problem of sampling from a stream of observations, such as network management, and database maintenance, a number of other approaches to stream sampling have been proposed, which imply various sampling distributions over the input. In some cases these are heuristic or hard to formalize as a precise distribution, but they may still have some desirable properties, such as providing unbiased estimates, or minimizing variance over certain queries. For example, the 'sample and hold' approach arises when we may see many occurrences of the same item, and we wish to estimate the number of occurrences. Each element is sampled uniformly, but then all subsequent occurrences

of elements currently in the sample are counted exactly [34, 42]. Other notable methods include Priority Sampling [31], Threshold Sampling [30], Variance Optimal (VarOpt) sampling [19], Adaptive Sample and Hold [34, 42], and Fair Sampling [29]. A common feature of these methods is that they apply in the model when elements only arrive, or equivalently, that weights of elements are positive. By contrast, most of the work in L_p sampling is relevant to the case when elements may depart as well as arrive, or more generally, when elements are subject to weight updates that can be both positive and negative.

Sampling based on hashing. As we see in subsequent sections, a key technical feature of L_p sampling algorithms is the use of randomly-chosen hash functions to ensure that when an element is seen multiple times, it is handled in the same way each time. This concept has also been used in a number of methods for sampling from streams of data, possibly observed at distributed locations. In its simplest form, this involves retaining information about elements whose hash values satisfy a certain property, to form the sample. For example, interpreting hash values as real numbers in the range $[0, 1]$, we might retain all elements whose hash value falls in $[0, 0.1]$ to obtain an (expected) 10% sampling rate. The notion of min-wise independent hash functions introduced above is used to sample elements uniformly over all permutations. This approach of keeping information about elements with the smallest hash values is referred to variously as bottom-k or k-mins sample [20, 72]. Since it is not influenced the associated weight of items, it is closest to L_0 sampling. The main difference in emphasis is again that L_0 sampling allows negative weights, which are not well-handled by the bottom-k and k-mins approaches: when the weight of a sampled element becomes zero due to the cancellation of positive and negative updates, the algorithm needs to recover a different sample element, which is not addressed in the k-mins/bottom-k work.

Sampling in Graph Streams. A notable special case in stream sampling is when the stream is considered to be formed of a stream of edges, which collectively define a graph. Here, the sampling problem is considered more challenging, since we typically are interesting in estimating properties of the sampled graph that go beyond single edges, such as connectivity properties or estimating the prevalence of small substructures (e.g. cliques). Consequently, much effort has been devoted to sampling methods in graphs, based on various exploration models such as random walks [67], the forest fire model [57], respondent sampling [43, 68] and many more [1]. L_p sampling has similarly found applications in graphs, most notably in answering connectivity questions (see Section 3.2). As with the previous examples, the main difference is that L_p sampling emphasises the dynamic case (edges can be added and removed), while other graph sampling results work on the arrivals-only model.

2 L_P SAMPLING ALGORITHMS

There are a number of algorithms which solve the L_p sampling problem with different properties and for different ranges of p. We begin with one that captures a very general case, and in subsequent subsections we describe algorithms that capture more specialised cases.

2.1 Canonical L_p sampling algorithm

In this section we describe the L_p sampling algorithm given in [49] for $p \in (0, 2]$.[‡] Other algorithms for this problem are somewhat similar in nature, and this primitive algorithm achieves a strong space/accuracy tradeoff. The basic logic of the sampler is as follows. The input vector $x = (x_1, \ldots, x_n)$ is scaled by random coefficients $u = (u_1^{-1/p}, \ldots, u_n^{-1/p})$, where each u_i is picked uniformly at random from $(0, 1)$. Let $z = (x_1 u_1^{-1/p}, \ldots, x_n u_n^{-1/p})$ be the scaled vector. Our sample will be determined by the largest elements of z. Due to the random choice of u, each entry in z has some chance to be largest, but those that correspond to larger values in x have a greater chance then those for smaller values. Formalizing this intuition allows us to show that this achieves the desired sampling distribution.

To this end, the important observation is $\Pr[u_i^{-1} \geq \tau] = 1/\tau$ and hence by setting the threshold τ to $\|x\|_p^p / |x_i|^p$, we get $\Pr[|z_i| \geq \|x\|_p] = |x_i|^p / \|x\|_p^p$. Therefore, assuming we have access to z and $\|x\|_p$, we look for the index $i \in [n]$ with weight $|z_i| \geq \|x\|_p$ and output the pair (i, z_i) as our sample. However, as we explain below, this idealized scenario can fail and produce an undesirable outcome for several reasons:

- First and foremost, it could be the case that no index reaches the threshold of $\|x\|_p$ and as result we end up with no samples. As we shall see, we can limit the probability of this event by repeating the sampler a large enough number of times and taking the first successful trial.
- Another source of error comes from the fact that, due to space limitations, we do not have full access to the scaled vector z and the norm $\|x\|_p$. To handle this problem, the algorithm keeps an approximation of the scaled vector using a Count-Sketch data structure and sets the threshold based on an approximate norm r, where $\|x\|_p \leq r \leq 2\|x\|_p$. The latter compromise does not affect the sampling distribution but the former could affect the outcome. As we will see in the proof of Lemma 2.1, with appropriate choice of parameters in the Count-Sketch data structures, the introduced error is not significant.
- Third and equally important, even if we had full access to z, it could be that for the choice of u there are several indices that reach the threshold of $r = O(\|x\|_p)$ and we end up with more than one 'winning' index. To limit the probability of this event, we raise the threshold of being selected as the chosen sample to a higher level (namely to $\frac{r}{\varepsilon^{1/p}}$). This limits the probability of multiple indices exceeding the threshold, at the expense of increasing the chance that none do.
- Finally to avoid having to store the scaling factors u, the algorithm selects the random coefficients u_i from a k-wise independent distribution for small k. The associated proof demonstrates that this is almost as good as using fully independent coefficients.

When these concerns have been addressed, the approach is appealing. The sampling is based entirely on data stored in sketches which are linear transformations of the input. This means that the algorithm is quite flexible, and can easily handle updates to the data in the form of positive and negative updates to components, multiple observers

[‡]In principle, the same approach could work for L_p sampling for $p > 2$. However, this parameter regime has attracted little study or application in the literature, so we do not address it.

Algorithm 1: The approximate L_p-sampler with success probability $\Theta(\varepsilon)$

Initialization.

- Set $k = 2\lceil \log(2/\varepsilon) \rceil$.
- Select $u_i \in [0,1]$ for $i \in [n]$ from a k-wise independent distribution.
- Set $m = \begin{cases} O(\log(1/\varepsilon)) & \text{if } p = 1 \\ O(\varepsilon^{-1} \log n) & \text{if } p = 2 \\ O(\varepsilon^{-\max(0,p-1)}) & \text{otherwise } (p \in (0,2) \setminus \{1\}) \end{cases}$

Processing the stream.

Update the required sketches to compute Count-Sketch$_m(z)$, $\hat{z}$, s and r where:

- $z \in \mathbb{R}^n$ is the vector x scaled by u
- $\hat{z}$ is the best m-sparse approximation of $z^* = $ Count-Sketch$_m(z)$
- s is a real with $\|z - \hat{z}\|_2 \le s \le 2\|z - \hat{z}\|_2$
- r is a real with $\|x\|_p \le r \le 2\|x\|_p$.

Output.

- Find i with $|z_i^*|$ maximal.
- If $s > \varepsilon^{1-1/p} m^{1/2} r$ or $|z_i^*| < \varepsilon^{-1/p} r$ output FAIL else output i as the sample.

combining their observations, and so on. We present a pseudo-code description of the algorithm in Algorithm 1. Note that the algorithm is described for success probability $\Theta(\varepsilon)$. We have the following lemma concerning Algorithm 1.

LEMMA 2.1. *Let $p \in (0,2]$. The probability that Algorithm 1 outputs the index $i \in [n]$ conditioned on a fixed value for $r \ge \|x\|_p^p$ is $(\varepsilon + O(\varepsilon^2))\frac{|x_i|^p}{r^p} + O(n^{-c})$.*

The proof of this lemma for the cases of $p \in (0,2)$ is shown in [49]. In this survey, we present of an extension of this proof to the case of $p = 2$. For the most part, the extended proof follows the same line of reasoning that is used for other values of p. For that reason we keep the parameter p in our proof and explicitly make a note of it when p is restricted to a fixed value.

Proof of Lemma 2.1 for the case of $p = 2$. We can neglect the low probability events of obtaining bad estimates of the various norms, as these simply affect the overall probability of success by negligible amounts. Hence, we can assume the values r and s are within the desired bounds. Namely,

$$\|x\|_p \le r \le 2\|x\|_p \tag{1}$$

$$\|z - \hat{z}\|_2 \le s \le 2\|z - \hat{z}\|_2 \le 20\,\mathrm{Err}_2^m(z) \tag{2}$$

$$|z_i^* - z_i| \le \mathrm{Err}_2^m(z)/m^{1/2} \text{ for all } i \in [n] \tag{3}$$

Before we proceed we remark that values s and r are easily computable using the sketching algorithm of Lemma 1.6. Now we enumerate the ways in which we could obtain a bad estimate, and bound their probability in each case.

(I) First we consider the case where $s > \varepsilon^{1-1/p}(m)^{1/2} r$ which causes the algorithm to output FAIL. Assume that the algorithm would otherwise have chosen to output the index i. To handle this case, we show that conditioned on $u_i = t$ where t is any fixed value, the probability that s goes beyond $\varepsilon^{1-1/p}(m)^{1/2} r$ is bounded by $O(\varepsilon + n^{-d})$. This

means no matter how large the coordinate i is, the tail error term s will be controlled with sufficiently high probability. To prove this, we bound the probability of the event $20\,\mathrm{Err}_2^m(z) \geq \varepsilon^{1-1/p}(m)^{1/2}\|x\|_2$ conditioned on $u_i = t$ for any fixed t. Together with the inequalities (1) and (2), this will prove our claim. For the rest of this part we set $p = 2$. This is where the proof differs from the cases $p \in (0, 2)$.

Recall that $\mathrm{Err}_2^m(z)$ is the L_2 norm of z after the largest m coordinates are set to zero. First we show that, with large enough probability, at most m coordinates of z are larger than a threshold $T = \varepsilon^{1/2}\|x\|_2$. The absence of these heavy coordinates from the error term $\mathrm{Err}_2^m(z)$ will enable us to use (a variant of) a Chernoff bound to put an upper bound on the mass of the non-heavy coordinates. Without loss of generality assume that the entries of z are indexed in increasing order of absolute value, so that the index i achieving the maximal value in z is $i = n$. For each $j \in [n-1]$ we define the indicator random variables z'_j as follows:

$$z'_j = \begin{cases} 1 & \text{if } |z_j| > T \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

Let $Z' = \sum_{j\in[n-1]} z'_j$. We have $\mathsf{E}[z'_j] = |x_j|^2/T^2$, from the definition of z_j and the previously calculated probability of exceeding any given threshold. Therefore, by linearity of expectation and using $T^2 = \varepsilon\|x\|_2^2$, we have that $\mathsf{E}[Z'] \leq \varepsilon^{-1}$. Since $m = \Omega(\log n/\varepsilon)$, from the concentration of Z' provided by k-wise independence (see Lemma 1.4) it follows $\Pr[Z' \geq m-1] = O(\varepsilon)$ as needed.

Now we define $z''_j = z_j^2/T^2$ for all $j \in [n-1]$. Let $Z'' = \sum_{j\in[n-m]} z''_j$. We have $\mathrm{Err}_2^m(z) = TZ''^{1/2}$ where $Z'' = \sum_{j\in[n-m]} z''_j$. Recall that we choose the indexing so that the largest entries of z are given the largest indices. Therefore we only need to prove an upper bound on Z''. First we analyze $\mathsf{E}[z''_i]$. A direct attempt to calculate this expectation runs into problems, since there is a small chance that u_i will be very small, which leads to a very high expectation. To avoid this, we condition on not having a very small value of u_i, the probability of which is correspondingly small. This ensures that the expectation does not grow too large. To this end let $\mathcal{M}'$ be the event that $u_i \geq n^{-5}$ for all $i \in [n-1]$. Note that $\Pr[\mathcal{M}'] \geq 1 - n^{-4}$. Now conditioned on $\mathcal{M}'$, we can bound $\mathsf{E}[z''_i]$. Namely,

$$\mathsf{E}[z''_i \mid \mathcal{M}'] = \frac{|x_i|^2}{T^2}\mathsf{E}[\tfrac{1}{u_i} \mid \mathcal{M}'] = \frac{|x_i|^2}{T^2}\left(\frac{1}{1-n^{-5}}\int_{n^{-5}}^{1}\frac{du}{u}\right) \leq \frac{10|x_i|^2}{T^2}\log n.$$

Thus, summing over all $i \in [n-m]$, we have $\mathsf{E}[Z'' \mid \mathcal{M}'] = O(\varepsilon^{-1}\log n)$.

Now let $\mathcal{M}$ be the event $Z' < m-1$. Observe that, conditioned on $\mathcal{M}$ which is the case with probability $1 - O(\varepsilon)$, we have $z''_j \in [0, 1]$ for all $j \in [n-m]$. Thus, putting all this together, we can bound the probability of Z'' exceeding $m = O(\varepsilon^{-1}\log n)$ by $O(\varepsilon + n^{-d})$ for some constant $d \geq 1$ using a variant of the Chernoff-style bound from Lemma 1.4 as we did for analyzing Z'. Then, with probability $1 - O(\varepsilon + n^{-d})$ (removing the conditioning on $\mathcal{M}$ and $\mathcal{M}'$), we have $\mathrm{Err}_2^m(z) \leq (\varepsilon m)^{1/2}\|x\|_2$, as required.

(II) Another undesirable event is that z_i exceeds the threshold $\varepsilon^{-1/p}r$ but z_i^* underestimates and as result i is not sampled. This case also leads to the algorithm outputting FAIL. A symmetrical event is when z_i is less than $\varepsilon^{-1/p}r$ while z_i^* overestimates and goes beyond the threshold. In this case, the algorithm does not necessarily output FAIL, but may report the index i incorrectly.

We bound the probability of both these cases together as one undesirable event. By inequality (3) and the assumption that $s \leq \varepsilon^{1-1/p}(m)^{1/2}r$, the additive error in estimating each z_i is at most $r\varepsilon^{1-1/p}$. Therefore this event could happen only when z_i falls in the interval

$$[r\varepsilon^{-1/p} - r\varepsilon^{1-1/p}, r\varepsilon^{-1/p} + r\varepsilon^{1-1/p}].$$

For this to hold u_i must be chosen from the interval

$$(\varepsilon^{-1/p} + \varepsilon^{1-1/p})^{-p}\frac{|x_i|^p}{r^p} \leq u_i \leq (\varepsilon^{-1/p} - \varepsilon^{1-1/p})^{-p}\frac{|x_i|^p}{r^p}.$$

We can analyze this range by studying

$$\begin{aligned}(\varepsilon^{-1/p} - \varepsilon^{1-1/p})^{-p} - (\varepsilon^{-1/p} + \varepsilon^{1-1/p})^{-p} &= \varepsilon((1-\varepsilon)^{-p} - (1+\varepsilon)^{-p}) \\ &= \varepsilon\left(\frac{(1+\varepsilon)^p - (1-\varepsilon)^p}{(1-\varepsilon^2)^p}\right) \\ &= \varepsilon\left(\frac{O(\varepsilon)}{\Omega(1)}\right) \\ &= O(\varepsilon^2)\end{aligned}$$

where we make use of the fact that $\varepsilon < \frac{1}{2}$, $0 < p \leq 2$.

Therefore the probability of this event is proportional to the size of the interval, which is hence $O(\varepsilon^2|x_i|^p/r^p)$ as required.

(III) A third bad outcome is that $|z_i| > \varepsilon^{-1/p}r$ but the algorithm outputs another coordinate $i' \neq i$ because $z^*_{i'} \geq z^*_i$. We show that the probability of multiple coordinates exceeding the threshold is small. Suppose $|z^*_{i'}| \geq \varepsilon^{-1/p}r$. This means $u_{i'} < \varepsilon|x_i|^p/r^p$ which happens with probability $O(\varepsilon|x_{i'}|^p/r^p)$. By the union bound the probability that such an index i' exists is $O(\varepsilon\|x\|_p^p/r^p) = O(\varepsilon)$. Using pairwise independence we can argue that the same bound holds after conditioning on $|z_i| > \varepsilon^{-1/p}r$.

We have covered all the possible bad outcomes. From the above arguments it follows that the total probability of the bad outcomes is bounded by $O(\varepsilon + n^{-d})$. This finishes the proof of the lemma. □

To boost the success probability we repeat Algorithm 1 $O(1/\varepsilon \log 1/\delta)$ times in parallel and take the outcome of the first successful trial. Considering the space requirements for maintaining the related sketches the following result holds.

Theorem 2.2. *For $\delta > 0$, $\varepsilon > 0$ and $p \in (0, 2]$ there is an $O(\varepsilon)$ relative error L_p sampler with failure probability at most δ and additive error $\Delta = O(n^{-c})$ where c is an arbitrary constant. The algorithm uses $O_p(\varepsilon^{-\max(1,p)}\log^2 n \log 1/\delta)$ space for $p \notin \{1, 2\}$ while for $p = 1$ and $p = 2$ the space usages are $O(\varepsilon^{-1}\log(1/\varepsilon)\log^2 n \log 1/\delta)$ and $O(\varepsilon^{-2}\log^3 n \log 1/\delta)$ respectively.*

Perfect L_p Sampling. Recently, Jayaram and Woodruff [46] extended this approach to show space efficient *perfect* L_p samplers. In other words, for $p \in (0, 2)$ they give L_p samplers with *zero* error in the sampling distribution ($\Delta = 0, \varepsilon = 0$) using only $O(\log^2 n \log(1/\delta))$ space. This matches the space complexity of the best approximations L_p in terms of dependence on n and δ while removing the dependence on ϵ. Considering the results of [51], this resolves the space complexity of L_p sampling for $p \in (0, 2)$. For

$p = 2$, the space complexity of [46] is $O(\log^3 n \log(1/\delta))$ which matches the space of the best approximate sampler without dependence on ε.

The overall logic of the Jayaram-Woodruff sampler follows the above canonical sampling algorithm. The input vector is multiplied by scaling factors drawn from a specific distribution and then the coordinate that achieves the maximum scaled value is picked as the chosen sample under the condition that certain (statistical) tests are passed. The chief point of deviation is to use the exponential distribution for the scaling factor, instead of the uniform distribution. Recall that t is an exponentially distributed random variable with mean $1/\lambda$ when $\Pr[t > x] < 1 - e^{\lambda x}$. The idea of using exponential distribution has previously shown fruitful in design of space efficient algorithms for estimating the frequency moments F_k for large k [6]. In addition to this, a different statistical test is applied to determine when to report the sampled item.

2.2 The L_0 Sampler

The above algorithm clearly doesn't work in the case $p = 0$ since (among other reasons) the scaling factors would be set as $u_i^{-1/0}$. Instead, we look to an algorithm that randomly selects subsets of possible coordinates to keep information on. Recall that, conditioned on no failure, a ε-relative error L_0 sampler with additive error Δ should return a non-zero coordinate with probability $(1 \pm \varepsilon)/\|x\|_0 + \Delta$. Almost all the existing L_0 sampler algorithms follow a similar pattern. To explain, consider the following ideal scenario. Assume $t = \Theta(\|x\|_0)$ is known beforehand and the algorithm is able to sample all possible coordinates independently at a rate of $O(1/t)$ producing the (implicit) vector x' where x'_i equals x_i for each sampled coordinate and zero elsewhere. In expectation we have $\|x'\|_0 = O(1)$ and further each non-zero coordinate of x has equal chance of appearing in x'. Given x', the problem of L_0 sampling reduces to a simpler problem of building a data structure to allow *recovery* of a vector with small Hamming weight. Formally speaking the recovery problem (also known as the randomized k-structure [41]) is defined as follows:

Definition 2.3. **(The Sparse Recovery Problem)**. Given parameter $s \in [n]$, the goal is to recover the vector $x \in \mathbb{R}^n$ defined by a stream of additions and subtractions on its coordinates. The algorithm should list all the non-zero coordinates of x (with their values) conditioned on $\|x\|_0 \leq s$ otherwise it should be reported that $\|x\|_0 > s$. An algorithm for this problem is allowed to err in the latter case where $\|x\|_0 > s$ and outputs $y \neq x$ with probability $\gamma < 1/4$.

There are a variety of solutions to this problem, which tradeoff the space required to perform the sparse recovery (the primary objective) against other factors such as the time to encode and decode, numerical stability, and bit complexity of the calculations. Using ideas from decoding Reed-Solomon codes over finite fields, it is possible to describe a deterministic procedure to efficiently recover an s-sparse vector from $2s$ linear measurements (see [58], section 12.9). An example of such a procedure (extended to the field of complex numbers) is given by Akçakaya and Tarokh [3] where the recovery time is $O(s^2)$, assuming constant time arithmetic operations. There are alternative methods with the same number of measurement but the recovery procedure involves the solution of a large linear program [15]. The L_0-sampler described in [23] uses a time-efficient randomized sparse recovery procedure using $O(s \log n \log(s/\gamma))$

Algorithm 2: A high-level description of the L_0 sampler

Initialization.

- Randomly choose $h : [n] \to [n^3]$ from a set of $O(\log n)$-wise independent functions.
- Initiate independent instances of the sparse recovery $\mathcal{R}_j(s, \gamma)$ for $j \in [O(\log n)]$ with $s = O(\log 1/\delta)$ and $\gamma = O(1)$ where the instance $\mathcal{R}_j$ processes a random sub-vector of x.

Processing the stream.
Given the update (i, u):

- For all $j \in [O(\log n)]$, if $h(i) \leq n^3/2^j$ pass the update (i, u) to $\mathcal{R}_j(s, \gamma)$.
- Update the sketch for computing $\|x\|_0 \leq r \leq 2\|x\|_0$.

Output.

- Let j be the smallest such that $2^j \geq r$.
- If the j-th sparse recovery outputs DENSE then output FAIL.
  Otherwise let x' be the recovered vector.
- If $x' = 0$ output FAIL.
- Else select the non-zero coordinate i that attains the minimum $h(i)$ and return (i, x'_i) as the sample.

bits of space where γ is the success probability of the procedure. The update time of the algorithm in [23] is $O(\log(s/\gamma))$ while the recovery time is $O(s \log(s/\gamma))$. The same space bound is shown by Ganguly [40]. The only problem here is the algorithm may output a wrong vector $y \neq x$ (even if the input vector is s-sparse.) To decrease the probability of such an outcome, we can add a simple randomized test to detect outputting $y \neq x$. For concreteness, we use this approach as the foundation of our presentation of a canonical and near optimal instantiation of an L_0 sampling algorithm. First, we state the result for sparse recovery.

LEMMA 2.4. *Given a vector x, defined by a stream of positive and negative updates, there is a randomized $O(s \log n \log(s/\gamma))$ space algorithm $\mathcal{Z}(s, \gamma)$ that outputs $y = x$ with probability $1 - \gamma$ provided that x is s-sparse, otherwise it either outputs DENSE or a vector $y \neq x$. In either case, the probability of returning $y \neq x$ is at most $O(n^{-c})$.*

PROOF. Let $\mathcal{R}'(s, \gamma)$ denote the sparse recovery procedure defined in [23]. Note that $\mathcal{R}'(s, \gamma)$ outputs x with probability γ if x is s-sparse. We supplement $\mathcal{R}'(s, \gamma)$ with a simple equality test. Over the prime field $(F_p)^n$, where p is a suitably large prime $p = O(n^c)$, we maintain a inner product "fingerprint" $r.x$ where $r = (r_0, r_0^2, \ldots, r_0^{n-1})$ and r_0 is a random element of the field. Let y be the output of $\mathcal{R}'(s, \gamma)$ in case it outputs a vector. If $r.x = r.y$ then we output y as the answer otherwise we output DENSE. The correctness of this procedure follows by considering the probability that r_0 is a solution to the polynomial $r.(x - y) = 0$. This is a polynomial of degree n, so the chance that a randomly chosen r_0 happens to be a root of this polynomial is n/p, which can be bounded as polynomially small in n, as required. The space cost is dominated by that of the sparse recovery procedure $\mathcal{R}'$. □

To remove the simplifying assumption of having prior knowledge of $O(\|x\|_0)$, we repeat the algorithm that down-samples the coordinates in parallel with different guesses $2, 4, 8, \ldots, 2^{\lfloor \log n \rfloor}$ for $\|x\|_0$. To sample a sub-vector, we select a k-wise independent random hash function $h : [n] \to [n^3]$ and use it in the following way: sampling the i-th coordinate with probability $1/t$ is simulated by the event $h(i) \leq n^3/t$. Last, the estimation procedure needs to determine the appropriate level of sampling to interrogate. One solution to this is that in parallel we run an algorithm to get a factor 2 approximation of $\|x\|_0$ with high probability (see the Initialization stage of Algorithm 2). In the end, we consider the instance based on the guess that is closest to the outcome of the ℓ_0-estimation procedure. A simpler alternative, at the expense of slightly increased costs, is to interrogate the data structure at each level of sampling, starting with the lowest probability of sampling, until a level with a non-zero vector is recovered. This slightly slows down the recovery step if many levels have to be inspected, and requires a modified proof to bound the error probability.

Studying the overall algorithm (presented as Algorithm 2), we have that if the sparse recovery outputs a zero vector or declares DENSE, the algorithm will output FAIL. Setting γ to a small enough constant and considering the fact that the sparse recovery procedure outputs a wrong vector with probability at most $O(n^{-c})$, this will result in a zero relative error L_0 sampler with additive error $O(n^{-c})$ using $O(\log^2 n)$ bits of space.

THEOREM 2.5. *There is a zero relative error L_0 sampler with additive error $\Delta = O(n^{-c})$ and δ failure probability that takes $O(\log^2 n \log 1/\delta \log\log 1/\delta)$ bits of space. The update time and recovery time of the algorithm is bounded by a polynomial in* $\log(n)$.

PROOF. The algorithm succeeds if the level j we choose to interrogate has at least one element from the input mapping there, and is not too dense, so that we can recover. We have chosen the parameter s so the probability of this event can be bounded with a Chernoff-like bound to occur with sufficiently high probability. It can then be argued that the sampling is uniform. Formally, in order to guarantee that $\Theta(1)$ non-zero coordinates are sampled with probability $1 - \delta$ we consider the level where $s = O(\log 1/\delta)$ non-zero coordinates are sampled in expectation.

The result then follows from Lemma 1.4, and the other properties are immediate. Further details of this approach can be found in [23]. □

We note that this algorithm compromises slightly on the space bound in order to provide a simpler algorithm and analysis. In particular, the $\log\log 1/\delta$ can be removed, at the expense of making multiple passes over the data structure and invoking a deeper mathematical result on the structure of random hypergraphs for the analysis. Specifically, one can use the "Invertible Bloom Filter" structure due to Eppstein and Goodrich to provide the sparse recovery procedure [33].

2.3 Insert-only Streams

Insert-only L_1 sampling. If we only allow positive updates, in the case of $p = 1$ the sampling problem becomes significantly more straightforward. For instance, the classical reservoir sampling [55] suggests a simple solution for L_1 sampling for insert-only streams. The basic algorithm maintains the sum of items weights as s, and chooses to replace the current sample with the new element of weight u with probability u/s.

Algorithm 3: L_1 sampler for positive updates

Initialization.

- Set the current sample coordinate $t = 0$.
- Set $v = 0$.
- Set the sum $s = 0$.

Processing the stream.
Given an update (i, u):

- Add u to s.
- With probability $\frac{u}{s}$ set $t = i$ and let $v = u$.
- Else, if i is the current sample add $2u$ to v.

Output.
Return t as the sampled coordinate and v as an estimate for x_t.

The probability of retaining an element at the end of the stream is exactly its weight divided by the sum of all weights. In Algorithm 3 we give pseudocode for a version of this algorithm which additionally provides an unbiased estimate for the sum of weights of all ocurrences of the sampled element within the whole stream.

LEMMA 2.6. *Fix $i \in [n]$. The Algorithm 3 without failure outputs i with probability exactly $x_i/\|x\|_1$. Conditioned on outputting i, $\mathsf{E}[v] = x_i$.*

PROOF. Let $u_1, \ldots, u_m$ be the sequence of updates without the label of the coordinate. Let $s_k = \sum_{j=1}^{k} u_j$ be the sum of the first k updates and let $u_{i_1}, \ldots, u_{i_r}$ be the updates corresponding to the i-th coordinate. By definition $s_m = \|x\|_1$ and $x_i = u_{i_1} + \ldots + u_{i_r}$. For $d \in [r]$, let E_{i_d} be the event where a replacement happens in the i_d-th step and no replacement happens afterwards. We have

$$\Pr[E_{i_d}] = \left(\frac{u_{i_d}}{s_{i_d}}\right)\left(\frac{s_{i_d+1} - u_{i_d+1}}{s_{i_d+1}}\right)\cdots\left(\frac{s_m - u_m}{s_m}\right) = \frac{u_{i_d}}{s_m}$$

Let F_i be the event of outputting i. Since the corresponding events $\{E_{i_d}\}$ are disjoint, it follows that

$$\Pr[F_i] = \sum_{d=1}^{r} \Pr[E_{i_d}] = \frac{u_{i_1} + \ldots + u_{i_r}}{s_m} = \frac{x_i}{\|x\|_1},$$

as desired.

For the second part, we have

$$\Pr[E_{j_d} \,|F_i] = \frac{\Pr[E_{id} \cap F_i]}{\Pr[F_i]} = \frac{u_{i_d}}{x_i}.$$

Consequently

$$\begin{aligned}\mathsf{E}[v \,|F_i] &= \sum_{d=1}^{r}\left(\frac{u_{i_d}}{x_i}\right)(u_{i_d} + 2u_{i_d+1} + \ldots + 2u_{i_r}) \\ &= \frac{1}{x_i}\sum_{d=1}^{r}(u_{i_d})(u_{i_d} + 2u_{i_d+1} + \ldots + 2u_{i_r})\end{aligned}$$

$$= \frac{1}{x_i}\left(\sum_{d=1}^{r} u_{i_d}^2 + 2\sum_{d=1}^{r}\sum_{j=d+1}^{r} u_{i_d}u_{i_j}\right)$$

$$= \frac{1}{x_i}\left(\sum_{d=1}^{r} u_{i_d}\right)^2 = \frac{x_i^2}{x_i} = x_i.$$

□

The space usage is $O(\log(mM))$ bits when M is the weight of the heaviest update.

Insert-only L_0 sampling. The case for L_0 sampling with an insert-only stream is somewhat similar. Again, we see each update in turn, and decide whether or not to make the new update (i, u) to be the sampled item based on a test. However, to achieve the desired sampling distribution, we base the test on i alone, and use a min-wise hash function to determine the sampled element. Specifically, we select the element that achieves the smallest hash value of the selected min-wise hash function as the sampled item. We can additionally maintain the sum of the associated weights u. This obtains an approximate L_0 sample using $O(\log(1/\varepsilon) + \log n)$ space, as a direct consequence of the properties of min-wise hashing as defined in Lemma 1.5.

Insert-only L_2 sampling. For the special case of $p = 2$ under unit-weight updates there is a perfect L_p sampler with constant failure probability and $O(\sqrt{n})$ space usage. Assuming the input is restricted to positive unit updates, the sampler works by taking $O(\sqrt{n})$ random indices from the input and checking if there is any identical pair among the samples. If the sampler does not find an identical pair it declares failure otherwise it outputs a random index that is found among the identical pairs. It is not hard to show that, conditioned on no failure, this sampler outputs the index i with probability $x_i^2/\|x_i\|_2^2$. A birthday paradox-like argument shows that the probability of failure is at most a constant. For streams with length $\Omega(n)$, this algorithm can be implemented in $O(1)$ update time. This gives the claimed result in Figure 1 [§].

Insert-only L_p sampling: general case. For other p values, the space cost is a little higher (see the state of the art in Figure 1), but still lower than for the general case with arbitrary insertions and deletions. In this case, we use the same exact procedure defined in Algorithm 3 except that to estimate the coordinates of the scaled vector z, we use the following recent result by Braverman *et al.* [11] instead of CountSketch.

LEMMA 2.7. *[11, Cor. 7] Given a sequence of positive integer updates defining the vector $f \in [M]^n$, there is a streaming algorithm that, with probability at least 3/4, returns a vector $\hat{f}$ where $|\hat{f}_i - f_i| \leq Err_2^c(f)/c^{1/2}$ using $O(c \log c \log n)$ bits of space.*

2.4 Sampling Multiple Items

Most applications require multiple samples of the data and as the number of samples increases, the efficiency of the sampling component itself can become a concern. The straightforward solution to obtain s samples is to run $O(s)$ independent instances of the sampler in parallel which introduces an $O(s)$ factor in the update time and space usage respectively. Some increase in space usage seems inevitable as we have to store

[§]This approach of sampling and looking for duplicates may be folklore; it was described to the authors by T. S. Jayram

each individual sample, however for better update times there are alternative solutions with faster processing times than the naive solution. In this direction, Barkay *et al.* [9] have shown an L_0 sampler with $O(\log \frac{s}{\delta})$ update time and $O(s \log \frac{s}{\delta})$ sample extraction time, at the expense of relaxing the independence requirement of the samples. The extracted samples are guaranteed to be $O(\log \frac{1}{\delta})$-wise independent which is sufficient for most applications. The algorithm is somewhat similar to the L_0 algorithm that we described in Section 2.2. It samples the domain $[n]$ at exponentially decreasing rates hence producing $O(\log n)$ sub-streams from the original stream. To save the space for keeping the random numbers (for generating the samples), the algorithm uses a $O(\log \frac{1}{\delta})$-wise independent source of randomness. For each level the algorithm maintains a fast s-sparse recovery sketch to recover s non-zero elements. Once an element is received it is processed by the corresponding levels and if necessary the associated s-sparse recovery sketches are updated. In the end of the stream a level with $L_0 = \Theta(s)$ is chosen and $\Theta(s)$ elements are successfully recovered which are guaranteed to be $O(\log \frac{1}{\delta})$-wise independent.

McGregor *et al.* [60] give a solution for fast L_p sampling where $p \in (0, 2]$ without the need to sacrifice the independence of the samples. Here we briefly describe the high level ideas of their solution. Assume we want to take s weighted samples from the vector $y = (|x_1|^p, \ldots, |x_n|^p)$. To accomplish this, in parallel we run two procedures. First we recover the heavy coordinates of y using a heavy hitter algorithm. These coordinates are more likely to be sampled, so it makes sense to recover them and store them individually. Second, we hash the coordinates into $O(s)$ buckets using a random hash function. (The latter is repeated in parallel for a fixed number of times.) The important observation here is that most buckets will be *good*, meaning that their total weight would be less than $O(1/s)\|y\|_1$. As a result with high probability at most $O(\log n)$ coordinates from each bucket will appear in a set of s samples. Consequently, we need only to maintain $O(\log n)$ local samplers for each bucket. To get one sample from the entire vector, we choose a random bucket from the good buckets (including the bucket set aside for the heavy hitters) relative to its total weight. Then from that bucket we draw a local sample and that will be the final sample. We refer the reader to [60] for full details.

2.5 Lower bounds

Communication complexity has shown to be very productive in obtaining space lower bounds for streaming algorithms. Since L_p samplers are streaming algorithms, space lower bounds for L_p samplers have been obtained from analyzing the related 'universal relation' communication game, which originally has been used to get circuit lower bounds [53, 54, 71]. In the universal relation communication game UR^n, two binary vectors are given to players Alice and Bob where Alice holds $x \in \{0, 1\}^n$ and Bob holds $y \in \{0, 1\}^n$. It is promised that $x \neq y$ and the goal is to output an index i such $x_i \neq y_i$. The deterministic communication complexity of the universal relation game is well studied and it is already three decades old (see [52, 71]). It is known that every deterministic protocol needs to transmit at least $n + 1$ bits to solve UR^n. On the other hand, there was not much interest in the randomized complexity of the universal relation until two decades later when a connection with L_p samplers was discovered in the work [49].

To understand how a communication lower bound for UR^n results in a space lower bound for L_p samplers, note that any approximate L_p sample (for $p \geq 0$) from the vector $z = x - y$ returns a coordinate i where $z_i \neq 0$ and hence it must be $x_i \neq y_i$. This means that Alice and Bob can use an L_p sampler algorithm to solve their universal relation problem with low failure rate, as follows. Assuming a shared source of randomness, Alice runs the L_p sampler over her updates of z (generated from the vector x) and sends the content of the memory to Bob where the execution of the algorithm is resumed on the updates generated from y. As result we get a one-way protocol for UR^n with failure probability δ where δ is the failure probability of the L_p sampler.

Authors in [49] have shown that any randomized one-way communication protocol for UR^n with constant probability of success requires a message of size at least $\Omega(\log^2 n)$ bits, hence proving a lower bound of $\Omega(\log^2 n)$ for L_p samplers with constant failure rate.

The lower bound for UR^n is established through a reduction from the Augmented Indexing problem. In an instance of Augmented Indexing $AIND_{m,k}$, Alice holds a vector $x \in [k]^m$ and Bob is given an arbitrary prefix of x, say $x_1 \ldots x_{i-1}$. Here k and m are positive integers. At the end, Bob should output x_i, the ith coordinate of x. It is known that in any one-way randomized protocol with shared randomness and $\delta < (1 - \frac{3}{2k})$ failure probability, Alice should transmit at least $\Omega((1-\delta)m \log k)$ bits [61]. The reduction from $AIND_{m,k}$ to UR^n works by setting $k = 2^t$ and $m = t$ where $t = \Theta(\log n)$ in the Augmented Indexing problem while δ is assumed to be a constant. Alice converts her input $x \in [2^t]^t$ to a binary vector x' of length n as follows. Each coordinate x_i is replaced with the concatenation of 2^{t-i} copies of a unit basis vector w of length 2^t, where the x_i-th coordinate of w is set to 1 while the rest are zero. Thus, the first coordinate x_1 is replaced with a binary vector of length $2^{t-1}2^t = 2^{2t-1}$. The second coordinate x_2 is replaced with a binary vector of length 2^{2t-2}, the third with 2^{2t-3}, and so on in an exponentially decreasing manner. The total length of x' is therefore $2^{2t} - 2^t$. We choose t so that $n = 2^{2t} - 2^t$. Bob performs the same kind of conversion on his input $x_1 \ldots x_{i-1}$ except that for missing numbers he replaces them with zero vectors of corresponding lengths, to obtain his vector y' of length n. Now if Alice and Bob run a universal relation protocol on x' and y' with constant probability of success, the result will reveal some index from a block, corresponding to index j where $i \leq j \leq m$. From the way we created x' from x, the position of the index reveals the value of x_j. If we can assume that the protocol outputs a uniformly random index from where x' and y' differ, then with constant probability Bob obtains a non-zero index in a block corresponding to the i-th coordinate of x, due to the greater number of copies of this part of the input. We then note that any randomized universal relation protocol can be transformed into a one where Bob does indeed sample a uniformly random position from the set of conflicting positions without extra communication.

Recently Kapralov *et al.* [51] have strengthened the lower bound for the universal relation to $\Omega(\min\{n, \log(1/\delta)\log^2 \frac{n}{\log(1/\delta)}\})$ for arbitrary failure probability δ and $p \in [0, 2)$. This nearly establishes the tightness of the upper bound given in [49] for constant ε.

3 APPLICATIONS

We have so far focused on how to achieve the desired L_p sampling distributions algorithmically, either exactly or approximately. In this section, we further motivate the importance of these techniques by studying some of the applications to which these methods have been put. In many cases, the problems tackled here had previously (explicitly or implicitly) been regarded as open, demonstrating the value of L_p sampling. By and large, these applications fall within the scope of Theoretical Computer Science, although many of these results may be considered foundational in data analytics and machine learning.

With respect to parameter p, the case of $p = 0$, i.e. L_0 sampling, has found the most applications. In almost every data stream scenario where distinct sampling over insert-only inputs is applicable, L_0 sampling can be used to extend results to handle item deletions as well. In the most prominent example, this has led to a variety of streaming algorithms for dynamic geometric problems. But most surprisingly, L_0 sampling has been utilized in deciding connectivity in graph streams and various other graph-related questions (see Figure 1 in [51] for a longer list of graph algorithms which make use of L_0 sampling). We briefly elaborate on some of these connections.

We have organized the applications into four main categories based on the nature of the problems and application areas: Matrix sampling and computation (Section 3.1), encompassing problems in regression and low-rank approximation; Graph Stream (Section 3.2), covering connectivity and subgraph counting; Dynamic geometric streams (Section 3.3), concerning spanning trees and width of a point set, and concluding with some miscellaneous applications (Section 3.4) on text and vector streams.

3.1 Matrix Computations

Matrix Sampling And Applications in Linear Algebra. The area of randomized numerical linear algebra (RNLA) is concerned with providing fast randomized and approximate solutions to large linear algebra problems. As the input size to these problems grows large, traditional methods which seek exact solutions do not scale well enough, and we are often content to tradeoff some error in the solution for a more feasible computation. Applications are often found for problems in machine learning and data analysis, such as regression. A variety of techniques are used here, in particular random projections for dimensionality reduction. Some of these techniques either directly or implicitly make use of ideas from L_p sampling.

Given a matrix A with dimension $n \times d$, the *row* L_p *sampling problem* asks us to sample a row index i with probability proportional to $\|A^{(i)}\|_p^p$, where $A^{(i)}$ denotes row i of A . This can be achieved by a simple reduction to vector L_p sampling, as follows. Consider linearizing matrix A to a vector a of dimension nd in some canonical fashion. We can apply L_p sampling to the vector a. When the sampled element a_j corresponds to an entry in row i of A, we consider i to have been sampled. It is immediate from the definition of L_p sampling that the probability of sampling row i is proportional to $\sum_{j=1}^{d} A_{i,j}^p = \|A^{(i)}\|_p^p$, as required. This enables subsequent applications in linear algebra:

Leverage Scores and Regression. The regression problem is a central problem in linear algebra. Give an $n \times d$ matrix A and n dimensional response vector y, we seek to find

a coefficient vector x to minimize $\|Ax - y\|_p$. The most common case of $p = 2$ is the well-known least-squares problem, where we aim to find x to minimize the squared error between Ax and y.

One way to understand regression is via the *leverage scores* of matrix A, defined as the squared Euclidean row-norms of the left-singular vectors of A. Equivalently, these can be expressed so that the leverage score of the ith row $A^{(i)}$ is $A^{(i)}(A^T A)^+(A^{(i)})^T$, where $^+$ denote the Moore-Penrose pseudoinverse. An early result on the approximate regression is due to Drineas, Mahoney and Muthukrishnan [28]. They prove that (with high probability) it suffices to solve a version of the problem where rows of A (and the corresponding entries of y) are sampled, according to a distribution over the lengths of the (left) singular vectors of the singular value decomposition (SVD) of A, i.e. the leverage scores. This distribution is essentially the L_2 row sampling distribution over A's singular vectors. Drineas *et al.* [27] show how to put this into a small space algorithm to efficiently perform this sampling in the streaming model of computation. The algorithm first takes an appropriate random projection of the data matrix, and forms the (pseudo)inverse of the projected data which can be composed with the L_p sampling applied to the input data to build a data structure which encodes the L_p sampling distribution applied to the (approximate) leverage scores.

Thus equipped with methods to sample according to the leverage score distribution, a number of other applications in linear algebra are possible. For example, Dasgupta *et al.* [26] extend to regression for other p values in the range $(0, 2]$, involving a step equivalent to L_p row sampling at the heart of the algorithm. Other applications of leverage scores and their generalizations that could appeal to L_p sampling includes the work of Alaiou and Mahoney [4] and Cohen, Musco and Musco [21]. Both papers make use of *ridge* leverage scores to aid in sampling columns for kernel ridge regression and low-rank approximation approximation (these modify the definition of leverage scores by adding a regularization term to the covariance matrix).

Distributed Low Rank Approximation. Approximating a large matrix by a small matrix has numerous applications in processing massive data sets [10]. One particular formulation of this problem has been studied by Frieze *et al.* [39] where the authors consider the problem of low-rank matrix approximation. In this problem, given a matrix A with dimension $n \times d$, the goal is to find a matrix D^* of rank at most k so that

$$||A - D^*||_F^2 \leq \min_{D, \text{rank}(D) \leq k} ||A - D||_F^2 + \varepsilon ||A||_F^2$$

where D^* can be represented compactly. Here $||A||_F^2$ is called the Frobenius norm of A and equals $\sum_{i,j} A_{i,j}^2$. The approach represents D^* in a factorized form: it samples a set of rows from A according to an appropriate distribution, and writes D as a product between A and the sampled rows. This compact description of D^* can be found in $\text{poly}(k, 1/\varepsilon)$ time (in particular, independent of n and d) under the assumption that one can sample the rows of A efficiently.

The required sampling distribution turns out to be the L_2 sampling distribution: the algorithm assumes access to an efficient sampler that samples the ith row $A^{(i)}$ with probability at least $c \frac{\|A^{(i)}\|_2^2}{\|A\|_F^2}$ for some constant $c \leq 1$.

Woodruff and Zhong [75] have studied this problem in the distributed server model where the global matrix A is distributed across s servers in the following fashion. The

server $i \in [s]$ holds a slice of the input, namely the matrix A^i where $A = f(\sum_{i=1}^{s} A^i)$. The function f is applied entry-wise to the matrix $\sum_{i=1}^{s} A^i$. In the simplest setting f is the identity function but it could be any polynomial function. Moreover, in Woodruff and Zhong's formulation the servers only communicate with the server which is responsible for computing the low-rank approximation of A. As we noted earlier, the algorithm of [39] computes the low-rank approximation from sampled rows of A. It follows that the main challenge in solving the distributed version is to efficiently sample a row of A according to the aforementioned distribution and with respect to a function of interest f. In particular for the softmax function (described below), Woodruff and Zhong give a solution that uses L_p sampling as a subroutine.

The softmax (Generalized Mean) function GM with parameter p is defined over n positive reals $x_1, \ldots, x_n$ as follows: $GM_p(x_1, \ldots, x_n) = (\frac{1}{n}(\sum x_i^p))^{\frac{1}{p}}$. For $p = 1$, this is simply the usual mean(.) function, while for large p, it approaches the max function. Now suppose each server $i \in [s]$ holds the local matrix $M^i \in \mathbb{R}^{n \times d}$. Here the global data A is defined as follows:

$$A_{i,j} = GM_p(|M^1_{i,j}|, \ldots, |M^s_{i,j}|).$$

To cast the problem in the terms that we described above, we can assume each server $t \in [s]$ raises his entries to the power of p and locally computes the matrix A^t where $A^t_{i,j} = \frac{1}{s}(M^t_{i,j})^p$. Now the global matrix is $A = f(\sum_{i=1}^{s} A^i)$ where f is the $x^{1/p}$ function. Recall that we want to sample the ith row of A with probability $c\frac{|A^{(i)}|^2}{||A||_F^2}$ for some constant $c \leq 1$. Given matrix A, we can sample a row by sampling a random entry of the matrix (after raising the entries to the power of 2) and then take the row that contains the sampled entry. Since A is not immediately available, we simulate this process by picking a sample entry of the matrix $\sum_{t=1}^{s} A^t$ according to an $L_{2/p}$ distribution over the entries of A^i's using an $L_{2/p}$ sampler. This produces the same outcome.

For the implementation, each server sends the CountSketch computation of its local $L_{2/p}$ sampler to the first sever. The first server extracts the final sample from the linear combination of the sketches. In total this requires only poly$(\log n, s)$ communication between the servers.

Estimating Cascaded Aggregates. Initial work in data stream processing considered how to estimate simple aggregates over the input (such as F_k). A more general question is what more complicated functions can be evaluated. For example, consider compound computations that first apply one function on subsets of the raw data, then aggregate partial results via additional functions. This is captured in the notion of "cascaded aggregates", and a canonical example are the cascaded frequency moments applied to a matrix of data. Given a $n \times d$ matrix A, the aggregate $F_k(F_p)(A)$ is defined as

$$\sum_{i=1}^{n} (\sum_{j=1}^{d} |A_{i,j}|^p)^k.$$

Monemizadeh and Woodruff [63] suggested an algorithm for approximating $F_k(F_p)$ that uses L_p sampling as subroutine. Their algorithm is a clever generalization of an earlier F_k estimation algorithm by Jayram and Woodruff [47]. Assuming $k \geq 1$, given a vector $b = (b_1, \ldots, b_n)$, the F_k-estimation algorithm in [47], in order to compute

$F_k(b) = \sum_{i=1}^{n} |b_i|^k$, takes $n^{1-1/k}\text{poly}(\varepsilon^{-1}\log(nd))$ samples from $[n]$ (coordinate IDs), where $i \in [n]$ is sampled with probability $|b_i|/\|b\|_1$. The algorithm in fact requires such samples from random subsets of coordinates of size 2^l, for each $l \in [\log n]$. To apply this algorithm to the cascaded aggregate function, coordinates of the vector b are seen as the F_p norms of the rows of the matrix A — namely, $b_i = \sum_{j=1}^{d} |A_{ij}|^p$. To be more precise, let $A(l)$ denote the random matrix derived by taking l random rows of A uniformly at random. The algorithm runs the sampler on the matrices $A(l)$, obtaining an entry in a given row i with probability

$$(1 \pm \varepsilon)\frac{\sum_{j=1}^{d} |A(l)_{i,j}|^p}{\sum_{i \in A(l)} \sum_{j=1}^{d} |A(l)_{i,j}|^p}.$$

The sampled row IDs are then fed to the F_k estimation algorithm of [47] to get an estimate of the cascaded aggregates. Algorithm 4 shows the main steps of the procedure.

Algorithm 4: Approximating $F_k(F_p)$ of matrix $A_{n\times d}$ by L_p sampling

Initialization.

- Set $T = n^{1-1/k}\text{poly}(\varepsilon^{-1}\log(nd))$.
- For $l \in [\log n]$, let $A(l)$ be a random subset of 2^l rows of A.

Processing the Stream.

- Run L_p-sampler algorithm T times in parallel on each $A(l)$.
- Feed row IDs of the samples from the $A(l)$ into the 1-pass F_k-algorithm of [47].

Output.
Return the output of the F_k-algorithm of [47] as the estimate for $F_k(F_p)$ of $A_{n\times d}$.

3.2 Graph Streams

Graph connectivity and spanning forests. Given a stream consisting of edge arrivals, computing a spanning forest of the end graph $G = (V, E)$ is an easy task. We start with an empty forest F. We include the newly arrived edge e in F iff including e does not introduce a cycle in F. Clearly at the end F will be a spanning forest of G and the space complexity of the algorithm is only $O(|V|\log|V|)$. In fact it has been shown that the space bound is tight [70]. However, the question becomes more challenging when edges can be *removed* as well as inserted to the graph. The algorithm which maintains a forest cannot repair the spanning forest in the presence of edge deletions unless one keeps track of all the edges in stream which requires space linear in $|E|$.

In a seminal work, Ahn *et al.* [2] developed an elegant solution for this problem that generates a linear sketch of size $O(|V|\log^3|V|)$ from the input stream using L_0-sampling as a subroutine. The constructed linear sketch is then used to test connectivity (and other properties as well) in an offline stage. Here we briefly mention some of the key ideas of the this algorithm. More details can be found in [2, 59].

The algorithm of Ahn *et al.* (here abbreviated as the AGM algorithm) simulates a simple recursive (offline) procedure, based on Boruvka's spanning tree algorithm. The

recursive procedure starts with a graph $G_0 = G(V, E)$ and through a series of rounds, shrinks the graph to produce smaller graphs $G_1, G_2, \ldots$ by picking edges to contract and unifying the nodes of contracted edges. The process is continued until every connected component is shrunk into a single isolated node. In more detail, in every round of the algorithm for each node $v \in V$ a random neighbor $u \in \Gamma(v)$ is chosen and then u and v are collapsed to form a super-node w whose neighbors are the union of the neighbors of u and v. Note that in one round, multiple nodes can collapse into a single super-node, if multiple edges incident on the same node are selected. The resulting graph on super-nodes is interpreted as a simple graph, i.e. multi-edges are treated as a single edge, and self-loops are ignored. In each round, we repeat the edge contraction process, and terminate the rounds when no edges remain. What remains at the end are the connected components of the original graph, each collapsed into an isolated super-node. Moreover, we can interpret the collection of edges selected for contraction as corresponding to a spanning forest of the original graph.

The crux of the AGM algorithm is to allow this offline procedure to be simulated based on a small amount of information captured from the input stream, via sampling, without having to revisit the input. We assume that the nodes are labeled with integers, so that for any edge (i, j) then either $i < j$ or $j < i$. The algorithm represents the adjacency list of each node v_i as a vector $\boldsymbol{a}^i \in \{-1, 0, 1\}^{\binom{n}{2}}$ so that $|\boldsymbol{a}^i_{(i,j)}| = 1 \iff (i, j) \in E$. If we encode these vectors via an L_0 sampler, then sampling an element provides a neighbor of v_i. This allows the first step of the above offline algorithm to be simulated. However, a naive encoding of the edge information would make it difficult to simulate subsequent steps of the edge-contraction algorithm. The key technical contribution of the AGM algorithm is to define an edge encoding so that we can combine information from a subset of nodes V' so that we can sample edges from the cut $(V', V \setminus V')$.

Given an edge $(v_i, v_j) \in E$, its representation is encoded in two different ways, once in the adjacency list of v_i, and once in the list for v_j. That is, the (i, j)-th coordinate in $\boldsymbol{a}^i$ is set to 1 if $i < j$, otherwise it is set to -1. All other coordinates are set to 0. Hence, in one of the two vectors the edge is encoded by a $+1$ and in the other it is encoded by a -1. Given a subset of nodes V', if we sum all the corresponding vectors then entries corresponding to edges between nodes $v_i \in V'$ and $v_j \in V'$ sum to zero and so cancel out. As a result, if we can sample from the non-zero entries of this aggregated vector for V', we obtain an edge from the cut. This is possible, due to the linear property of the L_0 sampling algorithms: we can obtain an L_0 sampler for the sum of a set of vectors by combining the L_0 samplers for each vector in the set.

So to simulate the whole process, the vectors $\boldsymbol{a}^i$ are defined (implicitly) from the input stream. For each vector $\boldsymbol{a}^i$ an L_0 sketch $sk(\boldsymbol{a}^i)$ is built as the stream is processed. The contraction process is applied after the whole stream is processed. Using the L_0 sketches, we choose a random neighbor for each node and contract the edges (v_i, v_j) by merging the corresponding sketches $sk(\boldsymbol{a}^i)$ and $sk(\boldsymbol{a}^i)$. One caution here is that reusing the same sketch for sampling in multiple rounds may lead to dependency issues (i.e. it is not clear how to argue that subsequent random choices are independent of earlier ones). To get around this problem, we build $O(\log n)$ independent sketches for each vertex v_i and each time we want to build a sketch of a super-node that contains v_i we use a fresh sketch $sk(\boldsymbol{a}^i)$ instead of reusing the old ones. We are guaranteed

that the algorithm concludes in $O(\log n)$ rounds with high probability, and so we have sufficient independent sketches with high probability. Since we can now simulate each round of the offline edge contraction algorithm using the sketches, the correctness of the AGM algorithm follows quite directly.

It is natural to ask whether this upper bound can be improved, in terms of the magnitude of the poly-logarithmic factor, either by modifications to the L_p sampling approach or otherwise. In particular, the algorithm does not require that an edge is sampled *uniformly* from the cut, only that some edge is chosen *arbitrarily* from the cut edges. It might seem that this relaxation could allow an improved space algorithm. However, this question has recently been settled in the negative: $O(|V| \log^3 |V|)$ space is necessary, as well as sufficient, as shown by Nelson and Yu [65]. Similar to lower bounds discussed in Section 2.5, the hardness is shown via the communication complexity of the Universal Relation problem, in the special case when Alice's input is a strict subset of Bob's.

Wedge sampling and triangle counting. A *wedge* in a graph $G = (V, E)$ is a simple path of length two. A wedge is called closed if its endpoints are connected via an edge (thus forming a *triangle*). A wedge that is not closed is called open. Because of its close relationship with triangles, wedge counting and sampling has gained attention in related studies [48, 60, 66]. Here we show an application of L_2 sampling to the problem of wedge sampling. We assume that the input stream is a sequence of edge insertions that defines a simple undirected graph.

A straightforward algorithm for sampling wedges is composed of two steps: first pick a random vertex $v \in V$ with probability $p_v = \binom{\deg(v)}{2}/W$. We call this the wedge distribution of the graph. Second, we sample two neighbors of v uniformly at random. Clearly this solution generates a sample wedge with the desired distribution but requires foreknowledge of the degree distribution, which is not feasible in one pass over the stream. To circumvent this problem, McGregor *et al.* [60] implement the solution in two passes using a polylog space algorithm. In the first pass a vertex v is sampled with probability $q_v = \deg(v)^2 / \sum_v \deg(v)^2$ using a L_2 sampler. Note that this is different from the wedge distribution but as we see in the proof of Lemma 3.1 as long as $m \geq 6n$ the result will be close enough. (The constant factor here is arbitrary.) In the subsequent pass, two independent random edges on v are picked. If the random edges are different, a sample wedge is returned otherwise the algorithm declares failure. Algorithm 5 describes the sampling procedure. The following lemma shows that with constant probability the algorithm successfully outputs a sample wedge from a distribution close to the wedge distribution.

Lemma 3.1. *Conditioned on $m \geq 6n$, there is a two pass algorithm that takes $O(\varepsilon^{-2} \log^2 n)$ space and outputs a random wedge where each wedge has the probability of $(1 \pm \varepsilon)\frac{1}{W}$ for being selected. The algorithm reports* fail *with probability at most* $1/3$.

Proof. We assume the L_2 sampler fails with probability at most $1/4$. Let F be the event that Algorithm 5 outputs fail. This occurs if the L_2 sampler fails or if we happen to sample the same edge twice. We bound this probability as follows:

$$\Pr[F] \quad \leq \quad \frac{1}{4} + \sum_{v \in V} \frac{1}{\deg(v)} \frac{\deg(v)^2}{\sum_{v \in V} \deg(v)^2} (1 \pm \varepsilon)$$

Algorithm 5: Two-pass algorithm for sampling a wedge using L_2 samplers

First pass:

- Run an ε-relative error L_2 sampler to sample a vertex v with probability proportional to $\deg(v)^2$.
- If the sampler returns no vertex, output FAIL.

Second pass:

- Sample edges e_1 and e_2 from $\{(v,u) : u \in \Gamma(v)\}$ with replacement.
- If $e_1 = e_2$ output FAIL otherwise output $(e_1 : e_2)$ as the sampled wedge.

$$\begin{aligned} &= \frac{1}{4} + \frac{\sum_{v\in V}\deg(v)}{\sum_{v\in V}\deg(v)^2}(1\pm\varepsilon) \\ &\leq \frac{1}{4} + \frac{2m}{4m^2/n}(1\pm\varepsilon) \leq 1/3. \end{aligned}$$

Let F' be the event that the algorithm outputs FAIL in the second pass. We show, conditioned on $\neg F'$, the probability of sampling any fixed wedge is within $(1\pm\varepsilon)1/W$. For this, let $\omega = (e_1 : e_2)$ be a fixed wedge centered on vertex u. Let E_ω be the event associated with sampling ω. Note that E_ω means that the algorithm has sampled u and $e_1 \neq e_2$. We have

$$\begin{aligned} \Pr[E_\omega|\neg F'] &= \frac{\Pr[E_\omega \wedge \neg F']}{\Pr[\neg F']} \\ &= \frac{\deg(u)^2}{\sum_{v\in V}\deg(v)^2}(1\pm\varepsilon)\left(\frac{2}{\deg(u)^2}\right) \Bigg/ \left(1 - \frac{\sum_{v\in V}\deg(v)}{\sum_{v\in V}\deg(v)^2}(1\pm\varepsilon)\right) \\ &= \frac{2(1\pm\varepsilon)}{\sum_{v\in V}\deg(v)(\deg(v)-1)} \\ &= (1\pm\varepsilon)\frac{1}{W} \end{aligned}$$

The space complexity is dominated by the space usage of the L_2 sampler. This finishes the proof of the lemma. □

It is worth mentioning that, conditioned on $W \geq m$, Madhav *et al.* [48] gives a 1-pass algorithm that outputs a random wedge using $O(\sqrt{n})$ space. The algorithm works by sampling edges uniformly at random and checking if two edges of a wedge have been sampled. By a similar argument to birthday paradox, one can show that sampling $O(m/\sqrt{W})$ edges is enough to find at least one wedge with constant probability. Simple information theoretic arguments show that this space bound is in fact optimal for one-pass algorithms. Hence, there is an exponential gap between what is possible in one-pass, and what is possible in two (using L_p sampling).

3.3 Dynamic Geometric Streams

Random sampling is widely used in the design of sublinear algorithms for geometric problems. Specifically in situations where the input data is described by a set of points $p_1, \ldots, p_n$ that lie in a typically low dimensional space such as $\mathbb{R}^d$, a small representative subset of points is often enough to calculate the cost of an approximate solution for the entire set. Various approximate summary structures for geometric

data such as ε-nets, ε-approximations and customized coresets can be constructed from a small sample of data [38, 44, 73]. This makes random sampling a powerful tool in tackling large geometric data. Moreover, thanks to L_0 sampling, sampling-based solutions have the potential to be extended to dynamic scenarios where the input data may include deletions of (already inserted) points. In fact, as we mentioned earlier in this survey, one of the first L_0 sampling algorithms [37] was designed to solve a dynamic geometric problem. (We shortly elaborate on this application here.) Following the work [37], variants of the L_0 sampling technique have been used in approximating the width of a point set [8], in construction of coresets for geometric optimizaiton problems [38] and k-means and k-median clustering [12, 36, 38].

To explain how L_0 is used within the context geometric problems, we briefly describe the dynamic geometric setting. We assume the stream consists a series of insertions and deletions of points from a low dimensional discrete space such as $X = \{0, \ldots, D\}^d$. Corresponding to each point in the space X (sometimes referred to as a cell), we have a component in an integer-valued vector $\boldsymbol{v} \in \mathbb{N}^{|X|}$. Whenever a point $p \in X$ is inserted (or deleted), the corresponding component in $\boldsymbol{v}$ is incremented (or decremented). Therefor $\boldsymbol{v}_i$ indicates the number of copies of the point from the i-th cell that are present in our input data. We assume each deletion corresponds to an insertion that has happened before, so the components of $\boldsymbol{v}$ never become negative. It should be clear that an L_0 sample from vector $\boldsymbol{v}$ gives us a random point from the stream (that is not deleted yet). Since vector $\boldsymbol{v}$ has $(D+1)^d$ components, the space usage of the L_0 sampler is bounded by $O(d \log^2 n)$ where $D = \text{poly}(n)$.

Now having explained the basic scenario, we begin with a simple toy problem. Suppose in the stream of points $p_1, \ldots, p_n$ we want to find a point that is farthest away from the origin. In other words, we would like to find p_{max} where

$$\|p_{max}\| = \max_i \{\|p_i\|\}$$

To make the problem even simpler, assume the points come from the one dimensional discrete interval $[0 \ldots D]$ where D is a power of 2. For insertion-only streams this problem is trivial. However for dynamic streams where inserted point could be deleted later, a space efficient solution is not so obvious. We show how we can apply L_0 sampling to find an approximate solution where at the end a point p_k is reported such that

$$\frac{1}{2}\|p_{max}\| \leq \|p_k\| \leq \|p_{max}\|.$$

The idea is to keep track of a set of sub-intervals of $[0 \ldots D]$. For each interval

$$I_1 = \left[\frac{D}{2} \ldots D\right], \quad I_2 = \left[\frac{D}{4} \ldots \frac{D}{2}\right), \quad I_3 = \left[\frac{D}{8} \ldots \frac{D}{4}\right), \quad \ldots \quad , \quad I_{\log D} = [1 \ldots 1]$$

we maintain an L_0 sampler with probability of failure δ. Let $\mathcal{S}_i$ be the sampler corresponding to the interval I_i. On the arrival of an update for point p (insertion or deletion), we update all the samplers $\mathcal{S}_i$ where $p \in I_i$. At the end of the stream, we take samples from each $\mathcal{S}_i$. For each i, $\mathcal{S}_i$ either declares FAIL or returns a sample point q_i. Let k be the smallest $i \in [\log D]$ such that $\mathcal{S}_i$ has not failed. The algorithm reports q_k as the solution.

We have the following claim. The proof follows from the description of the algorithm.

Claim 3.2. *Let I_m be the smallest interval that contains p_{max}. With probability $1-\delta$, $\mathcal{S}_m$ returns a sample point. Moreover, conditioned on $\mathcal{S}_m$ returning a sample, we have $\frac{1}{2}\|p_{max}\| \leq \|q_m\| \leq \|p_{max}\|$.*

Since we maintain $\log D$ samplers, the space usage is bounded by $O(\log D \log^2 n \log(1/\delta))$. It is fairly straightforward to extend this algorithm to a higher dimension. For instance in the plane, we just need to maintain L_0 samplers for a set of nested quadrants of $[0 \ldots D]^2$ and at the end report a point based on the outcome of the samplers. The cost remains $O(\log D)$ samplers (with some geometric calculations needed to determine which samplers to represent each arriving point in). Similarly, it is straightforward to provide a relative error approximation, by increasing the number of samplers to $O(\frac{1}{\epsilon}\log D)$.

Width of Points. A similar idea to what we just described has been used in the work by Andoni and Nguyen [8] to estimate the width of a set of dynamic points using L_0 sampling. The width of a set of points $S = \{p_1, \ldots, p_n\}$ is the minimal distance between a pair of hyperplanes that sandwiches S. When restricted to a certain direction, the directional width of S with respect to the unit vector $\boldsymbol{u}$ denote by $W_{\boldsymbol{u}}(S)$ is the minimal distance between a pair of hyperplanes perpendicular to the vector $\boldsymbol{u}$ that sandwiches S. Suppose q is a point on a hyperplane that is in the middle of the two sandwiching hyperplanes. Then the directional width with respects to the direction $\boldsymbol{u}$ and the middle point $\boldsymbol{q}$ denoted by $W_{\boldsymbol{u},t}(S)$ equals

$$2 \max_i |\boldsymbol{u}.\boldsymbol{p}_i - \boldsymbol{v}.\boldsymbol{q}|$$

Here $\boldsymbol{x}.\boldsymbol{y}$ is the dot product of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Imagining each $|\boldsymbol{u}.\boldsymbol{p}_i - \boldsymbol{v}.\boldsymbol{q}|$ is like a point on the one dimensional line, the problem translates to the toy problem that we discussed above. Of course the challenge in estimating the width is to cover all directions $\boldsymbol{u}$ and the middle points $\boldsymbol{q}$, which forms the main focus of [8]. Once the reduction has been made, this particular subproblem is easily solved by L_0 sampling.

Minimum Euclidean Spanning Tree. The work by Frahling *et al.* [37] is based on a more sophisticated use of L_0 sampling. Their goal is to maintain the cost of an approximate minimum spanning tree over a set of points in a low dimensional space. The stream consists of series of insertions and deletions of points. Note that here the edges of the underlying graph are implicit; the weight of each edge is proportional to the distance between the corresponding endpoints. Using a well-known reduction described by Chazelle *et al.* [18], the problem of estimating the cost of the minimum spanning tree is reduced to estimating the number of connected components. Since we are concerned about the application of L_0 sampling, we omit the details of the reduction and highlight the part where the sampler is used.

Let c^t denote the number of connected components of the underlying geometric graph G^t where G^t contains all the edges with weight at most t. Estimates of c^t for various values of t are then used to approximate the cost of the minimum spanning tree. To estimate c^t, Frahling *et al.* [37] use the randomized algorithm of [18]. This algorithm estimates the number connected components by performing BFS-like explorations starting from a random set of vertices. Since the depth of the BFS explorations is limited, to simulate the algorithm in the streaming context, it is enough to sample a set of points (starting vertices) and obtain all the points that are located within a certain

radius of the sampled points. When the input is dynamic, the sub-problem of graph sampling is implemented using L_0 sampling. To obtain the neighborhood information of each sampled point, Frahling *et al.* [37] use a modified vector $\boldsymbol{v}^*$ (instead of $\boldsymbol{v}$) where each $\boldsymbol{v}_i^*$ is a large D^d-ary number that encodes the neighborhood information of the corresponding cell within a certain radius. Whenever an update happens in a cell, all the corresponding components associated with neighboring cells are also updated. As result, when we sample a component of $\boldsymbol{v}^*$, we also obtain information regarding the points that lie within a certain radius of of the sample. In a typical setting, the large D^d-ary numbers increase the space usage of the L_0 sampler from $O(d \log^2 n)$ to $O(d^2 \log^2 n)$.

3.4 Miscellaneous streaming problems

Finding Duplicates. The duplicate finding problem is a somewhat abstract problem introduced to demonstrate of the extreme challenges that arise in data stream processing. Given a data stream of length $n+1$ over the alphabet $[n]$, we are guaranteed that there must be (at least) one character that appears twice in the stream; the problem is to output some $a \in [n]$ that does indeed appear more than once. In a generalization of the problem, we are given an input stream of length $n - s \leq n$ over the alphabet $[n]$. The key to finding a solution is to reduce the question to a sampling one, where a sufficient amount of the probability mass is associated with samples that correspond to duplicate elements. For this problem, Jowhari *et al.* [49] gave an $O(s \log n + \log^2 n)$ bits algorithm using L_1-sampling. Here we mention the main ideas of the algorithm for the special case where the length of the stream is $n+1$ in the proof of the following lemma.

Lemma 3.3. *There is a one-pass $O(\log^2 n \log(\frac{1}{\delta}))$ space algorithm such that given a stream of length $n+1$ over the alphabet $[n]$ returns an $i \in [n]$ or declare FAIL. The probability of declaring failure is at most δ while the probability of outputting a non-duplicate is $O(n^{-c})$ for a constant c.*

Proof. In the following we describe how the problem is reduced to L_1 sampling. We define the vector $x \in \{-n, \ldots, n\}^n$ based on the input stream as follows. The coordinate x_i equals the number of times the character i appears in the stream after subtracting 1 from it. So if character i appears 4 times in the stream the coordinate x_i would equal 3. Now this vector has some features that are important for our purpose. First, the strictly positive coordinates in x represent the duplicate characters. $x_i = -1$ means that i did not appear in x while $x_i = 0$ indicates that i is non-duplicate character in the stream. Importantly for the purpose of finding duplicates, the total weight of positive coordinates is bigger than the total weight of the negative ones. That is,

$$\left(\sum_{i:x_i>0} x_i\right) > \left(\sum_{i:x_i<0} |x_i|\right).$$

Putting these facts together, it means that if we take samples from x according to its L_1 distribution, there is a good chance (at least $1/2$) that we hit upon a strictly positive coordinate in x, in other words we find a duplicate character.

Taking L_1 samples from x is easy. To do this, the algorithm treats the input stream as updates to an (initially zero) vector which at the end becomes the vector x. The

algorithm then subtracts 1 from each coordinate of x before carrying out the sampling procedure. If the L_1 sampler returns a negative coordinate or returns no samples we declare FAIL otherwise with high confidence we can claim that the returned coordinate is a real duplicate. To boost the probability of success we can repeat the sampling in parallel and take the first successful trial. □

Estimating F_p for $p > 2$. As we observed in Section 2, existing L_p samplers rely on getting an approximation of $\|x\|_p$ to get a sample from the stream. As we see in this section, these samplers themselves can be applied to the problem of approximating $\|x\|_p$ for $p > 2$ when the stream is restricted to positive updates. More precisely we can approximate F_p for $p > 2$ using L_2 samplers. Coppersmith and Kumar [22] outlined the approach to estimating F_p for $p > 2$ based on sampling according to what we would now describe as the L_2 distribution, and scaling up the estimated frequency of the sampled element to provide an estimator for F_p. However, at the time of publication, no such samplers were known. Monemizadeh and Woodruff [63] subsequently took this outline and instantiated it with their first construction of an L_2 sampler. In more detail, the algorithm works by taking $T = O(n^{1-2/k}k^2/\varepsilon^2)$ L_2 samples of the input vector. A high-level description of the algorithm is written out as Algorithm 6.

Algorithm 6: Estimating F_k using L_2 sampling

Initialization.

- Set $\varepsilon'' = \varepsilon/(4k)$
- Set $T = O(n^{1-2/k}/\varepsilon''^2)$.

Processing the stream.

- Run an ε-relative error L_2-sampler $4T$ times in parallel.
- In parallel, run an algorithm to approximate $F_2(x)$ within $1 + \varepsilon/4$ factor with success probability $7/8$.

Output.

- Let the first T output frequencies from the L_2 sampler be $a_{i_1}, a_{i_2}, \ldots, a_{i_T}$.
- If more than $3T$ of the L_2-samplers output FAIL, declare FAIL.
- Denote the output of the F_2 estimation by $\tilde{F}_2$.
- Return $G = \frac{\tilde{F}_2}{T}\sum_{j=1}^{T}|a_{i_j}|^{k-2}$ as the estimate for F_k.

THEOREM 3.4. *For $k > 2$, there is a $O(k^2\varepsilon^{-4}n^{1-2/k}\log^2 n)$ space algorithm for approximating F_k within $1 + \varepsilon$ factor.*

PROOF. To simplify the notation, we write F_2 and F_k respectively to denote the second and the k-th frequency moment of the underlying vector. In the following we assume the estimation procedure does not declare FAIL so that the algorithm gathers the required number of samples. We also assume that the estimate of F_2 succeeds in meeting its error bound of $\varepsilon F_2/4$. For any $j \in [T]$, we have

$$\mathsf{E}[|a_{i_j}|^{k-2}] \quad = \quad \sum_{i=1}^{n}(1 \pm \varepsilon'')\frac{|a_i|^2}{F_2}(|a_i|(1 \pm \varepsilon''))^{k-2}$$

$$= \quad (1 \pm \varepsilon'')^{k-1} \frac{F_k}{F_2}.$$

Consequently, by our choice of ε'', we have

$$\mathsf{E}[G] = T(1 \pm \varepsilon'')^{k-1}(1 \pm \varepsilon/4)\frac{F_k F_2}{T F_2} = (1 \pm \varepsilon/2)F_k.$$

Since the samples are independent and $\tilde{F}_2 < 2F_2$, we have

$$\begin{aligned}
\mathsf{Var}[G] &= \frac{4F_2^2}{T^2}\sum_{i=1}^{T}\mathsf{Var}[|a_{i_j}|^{k-2}] \\
&= \frac{4F_2^2}{T^2}T\sum_{i=1}^{n}(1 \pm \varepsilon'')\frac{|a_i|^2}{F_2}(|a_i|(1 \pm \varepsilon''))^{2k-4} \\
&\leq \frac{4F_2^2 e^{\varepsilon(2k-4)/4k}}{T}\sum_{i=1}^{n}\left(\frac{2|a_i|^2}{F_2}\right)|a_i|^{2k-4} \\
&= O\left(\frac{F_2 F_{2k-2}}{T}\right) = O\left(\frac{n^{1-2/k}F_k^2}{T}\right) = O(\varepsilon^2 F_k^2)
\end{aligned}$$

Here the last statement follows from Hölder's inequality (See [63] for details). Using Chebyshev's inequality with appropriate rescaling of constants, we get

$$\Pr[|G - \mathsf{E}[G]| \geq \frac{\varepsilon}{4}\mathsf{E}[G]] \leq 1/16.$$

That is, there is constant probability of obtaining an estimate of F_k within the desired accuracy bound. To analyze the space complexity, the space usage of the algorithm is dominated by the total space needed to run the L_2 samplers. Since we are only concerned with positive updates, each sampler takes only $O(1/\varepsilon^2 \log^2 n)$ space (see Section 2.3) . This proves the claimed space bound. □

4 CONCLUDING REMARKS

The notion of L_p sampling and the existence of efficient L_p samplers, have enabled a number of applications and solutions to previously open problems in algorithms. These have addressed data which is statistical, graph structured and geometric in nature. New problems are being addressed by these techniques either directly, or by methods inspired by L_p sampler algorithms. We anticipate further advances using these approaches to be developed in the coming years, and new efforts to make these algorithms more practical for implementation.

The sampling algorithms themselves are relatively simple and can be implemented building on implementations of sketches and sparse-recovery algorithms. However, the space required to provide sufficiently strong guarantees of success becomes large, due to the number of repetitions required. Lower bounds suggest that the asymptotic dependence on parameters ε and δ is unavoidable, but there may be scope to improve constant factors, or identify additional special cases for which more efficient samplers exist. Such improvements to the practicality of L_p samplers is an important open problem which could lead to greater applicability of these algorithms.

ACKNOWLEDGMENTS

We thank the anonymous ACM Surveys reviewers for many helpful comments and suggestions which have improved the presentation and coverage of this survey.

REFERENCES

[1] Ahmed, N. K., Neville, J., and Kompella, R. R. Network sampling: From static to streaming graphs. *TKDD 8*, 2 (2013), 7:1–7:56.

[2] Ahn, K. J., Guha, S., and McGregor, A. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms* (2012), pp. 459–467.

[3] Akçakaya, M., and Tarokh, V. A frame construction and a universal distortion bound for sparse representations. *IEEE Trans. Signal Processing 56*, 6 (2008), 2443–2450.

[4] Alaoui, A. E., and Mahoney, M. W. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (2015), pp. 775–783.

[5] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci. 58*, 1 (1999), 137–147.

[6] Andoni, A. High frequency moments via max-stability. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017* (2017), pp. 6364–6368.

[7] Andoni, A., Krauthgamer, R., and Onak, K. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science* (2011), pp. 363–372.

[8] Andoni, A., and Nguyên, H. L. Width of points in the streaming model. *ACM Trans. Algorithms 12*, 1 (2016), 5:1–5:10.

[9] Barkay, N., Porat, E., and Shalem, B. Efficient sampling of non-strict turnstile data streams. In *Fundamentals of Computation Theory* (2013), pp. 48–59.

[10] Blum, A., Hopcroft, J., and Kannan, R. *Foundations of data science.* http://www.cs.cornell.edu/jeh/book.pdf, 2018.

[11] Braverman, V., Chestnut, S. R., Ivkin, N., Nelson, J., Wang, Z., and Woodruff, D. P. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (2017).

[12] Braverman, V., Frahling, G., Lang, H., Sohler, C., and Yang, L. F. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (2017), pp. 576–585.

[13] Broder, A. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences (SEQUENCES'97)* (1997), pp. 21–29.

[14] Broder, A., Charikar, M., Frieze, A., and Mitzenmacher, M. Min-wise independent permutations. In *ACM Symposium on Theory of Computing* (1998), pp. 327–336.

[15] Candès, E. J., Rudelson, M., Tao, T., and Vershynin, R. Error correction via linear programming. In *46th Annual IEEE Symposium on Foundations of Computer Science* (2005), pp. 295–308.

[16] Carter, J. L., and Wegman, M. N. Universal classes of hash functions. *Journal of Computer and System Sciences 18*, 2 (1979), 143–154.

[17] Charikar, M., Chen, K. C., and Farach-Colton, M. Finding frequent items in data streams. *Theor. Comput. Sci. 312*, 1 (2004), 3–15.

[18] Chazelle, B., Rubinfeld, R., and Trevisan, L. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput. 34*, 6 (2005), 1370–1379.

[19] Cohen, E., Duffield, N., Kaplan, H., Lund, C., and Thorup, M. Sketching unaggregated data streams for subpopulation-size queries. In *Proc. 26th ACM Symp. on Principles of Database Systems (PODS)* (2007).

[20] Cohen, E., and Kaplan, H. Bottom-k sketches: better and more efficient estimation of aggregates. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS* (2007), pp. 353–354.

[21] Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on*

Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19 (2017), pp. 1758–1777.

[22] Coppersmith, D., and Kumar, R. An improved data stream algorithm for frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2004), pp. 151–156.

[23] Cormode, G., and Firmani, D. A unifying framework for l0-sampling algorithms. *Distributed and Parallel Databases 32*, 3 (2014), 315–335.

[24] Cormode, G., and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms 55*, 1 (2005), 58–75.

[25] Cormode, G., Muthukrishnan, S., and Rozenbaum, I. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st International Conference on Very Large Data Bases* (2005), pp. 25–36.

[26] Dasgupta, A., Drineas, P., Harb, B., Kumar, R., and Mahoney, M. W. Sampling algorithms and coresets for ℓ_p regression. *SIAM J. Comput. 38*, 5 (2009), 2060–2078.

[27] Drineas, P., Magdon-Ismail, M., Mahoney, M. W., and Woodruff, D. P. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research 13* (2012), 3475–3506.

[28] Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Sampling algorithms for l_2 regression and applications. In *ACM-SIAM Symposium on Discrete Algorithms* (2006), pp. 1127–1136.

[29] Duffield, N. Fair sampling across network flow measurements. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (2012), SIGMETRICS '12, ACM, pp. 367–378.

[30] Duffield, N., Lund, C., and Thorup, M. Learn more, sample less: control of volume and variance in network measurements. *IEEE Transactions on Information Theory 51*, 5 (2005), 1756–1775.

[31] Duffield, N., Lund, C., and Thorup, M. Priority sampling for estimation of arbitrary subset sums. *J. ACM 54*, 6 (December, 2007), Article 32. Announced at SIGMETRICS'04.

[32] Efraimidis, P. S., and Spirakis, P. G. Weighted random sampling with a reservoir. *Inf. Process. Lett. 97* (2006), 181–185.

[33] Eppstein, D., and Goodrich, M. T. Space-efficient straggler identification in round-trip data streams via newton's identities and invertible bloom filters. In *Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007, Proceedings* (2007), pp. 637–648.

[34] Estan, C., and Varghese, G. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM* (2002), vol. 32, 4 of *Computer Communication Review*, pp. 323–338.

[35] Fan, C., Muller, M., and Rezucha, I. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. Amer. Stat. Assoc. 57* (1962), 387–402.

[36] Frahling, G. *Algorithms for dynamic geometric data streams.* PhD thesis, University of Paderborn, Germany, 2006.

[37] Frahling, G., Indyk, P., and Sohler, C. Sampling in dynamic data streams and applications. In *Proceedings of the 21st ACM Symposium on Computational Geometry* (2005), pp. 142–149.

[38] Frahling, G., and Sohler, C. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005* (2005), pp. 209–217.

[39] Frieze, A. M., Kannan, R., and Vempala, S. Fast monte-carlo algorithms for finding low-rank approximations. *J. ACM 51*, 6 (2004), 1025–1041.

[40] Ganguly, S. Counting distinct items over update streams. *Theor. Comput. Sci. 378*, 3 (2007), 211–222.

[41] Ganguly, S., and Majumder, A. Deterministic k-set structure. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (2006), pp. 280–289.

[42] Gibbons, P., and Matias, Y. New sampling-based summary statistics for improving approximate query answers. In *ACM SIGMOD International Conference on Management of Data* (1998), pp. 331–342.

[43] Goel, S., and Salganik, M. J. Respondent-driven sampling as markov chain monte carlo. *Statistics in Medicine 28*, 17 (2009), 2202–2229.

[44] Haussler, D., and Welzl, E. epsilon-nets and simplex range queries. *Discrete & Computational Geometry 2* (1987), 127–151.

[45] Indyk, P. A small approximately min-wise independent family of hash functions. *J. Algorithms 38*, 1 (2001), 84–90.

[46] Jayaram, R., and Woodruff, D. P. Perfect $\mathrm{l_p}$ sampling in a data stream. *FOCS 2018* (2018).

[47] Jayram, T. S., and Woodruff, D. P. The data stream space complexity of cascaded norms. In *50th Annual IEEE Symposium on Foundations of Computer Science,* (2009), pp. 765–774.

[48] Jha, M., Seshadhri, C., and Pinar, A. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *TKDD 9*, 3 (2015), 15:1–15:21.

[49] Jowhari, H., Saglam, M., and Tardos, G. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2011), pp. 49–58.

[50] Kane, D. M., Nelson, J., and Woodruff, D. P. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA* (2010), pp. 1161–1178.

[51] Kapralov, M., Nelson, J., Pachocki, J., Wang, Z., Woodruff, D. P., and Yahyazadeh, M. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. *CoRR abs/1704.00633* (2017).

[52] Karchmer, M. *Communication complexity - a new approach to circuit depth.* MIT Press, 1989.

[53] Karchmer, M., Raz, R., and Wigderson, A. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity 5*, 3/4 (1995), 191–204.

[54] Karchmer, M., and Wigderson, A. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA* (1988), pp. 539–550.

[55] Knuth, D. E. *The Art of Computer Programming, Volume 2: (2nd Ed.) Seminumerical algorithms.* Addison Wesley Longman Publishing Co., Inc., 1969.

[56] Knuth, D. E. *The Art of Computer Programming, Vol 3, Sorting and Searching*, 2nd ed. Addison-Wesley, 1998.

[57] Leskovec, J., and Faloutsos, C. Sampling from large graphs. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2006).

[58] MacWilliams, F. J., and Sloane, N. J. A. *The theory of error correcting codes.* North-Holland mathematical library. North-Holland Pub. Co. New York, Amsterdam, New York, 1977. Includes index.

[59] McGregor, A. Graph stream algorithms: a survey. *SIGMOD Record 43*, 1 (2014), 9–20.

[60] McGregor, A., Vorotnikova, S., and Vu, H. T. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (2016), pp. 401–411.

[61] Miltersen, P. B., Nisan, N., Safra, S., and Wigderson, A. On data structures and asymmetric communication complexity. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA* (1995), pp. 103–111.

[62] Mitzenmacher, M., and Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[63] Monemizadeh, M., and Woodruff, D. P. 1-pass relative-error l_p-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (2010), pp. 1143–1160.

[64] Motwani, R., and Raghavan, P. *Randomized Algorithms.* Cambridge University Press, 1995.

[65] Nelson, J., and Yu, H. Optimal lower bounds for distributed and streaming spanning forest computation. *CoRR abs/1807.05135* (2018).

[66] Pavan, A., Tangwongsan, K., Tirthapura, S., and Wu, K. Counting and sampling triangles from a graph stream. *PVLDB 6*, 14 (2013), 1870–1881.

[67] Ribeiro, B., and Towsley, D. Estimating and sampling graphs with multidimensional random walks. In *IMC* (2010), pp. 390–403.

[68] Salganik, M. J., and Heckathorn, D. D. Sampling and estimation in hidden populations using respondent-driven sampling. *Sociological Methodology 34*, 1 (2004), 193–240.

[69] Schmidt, J. P., Siegel, A., and Srinivasan, A. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math. 8*, 2 (1995), 223–250.

[70] Sun, X., and Woodruff, D. P. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)* (2015), pp. 435–448.

[71] Tardos, G., and Zwick, U. The communication complexity of the universal relation. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity* (1997), pp. 247–259.

[72] Thorup, M. Bottom-k and priority sampling, set similarity and subset sums with minimal independence.

In *Symposium on Theory of Computing Conference* (2013), pp. 371–380.

[73] Vapnik, V. N., and Chervonenkis, A. Y. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl. 16* (1971), 264—-280.

[74] Vitter, J. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1 (1985), 37–57.

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