Data Streams with Bounded Deletions

Extended Abstract*

Rajesh Jayaram
Carnegie Mellon University
Pittsburgh, PA
rkjayara@cs.cmu.edu

David P. Woodruff
Carnegie Mellon University
Pittsburgh, PA
dwoodruf@cs.cmu.edu

ABSTRACT

Two prevalent models in the data stream literature are the insertion-only and turnstile models. Unfortunately, many important streaming problems require a $\Theta(\log(n))$ multiplicative factor more space for turnstile streams than for insertion-only streams. This complexity gap often arises because the underlying frequency vector f is very close to 0, after accounting for all insertions and deletions to items. Signal detection in such streams is difficult, given the large number of deletions.

In this work, we propose an intermediate model which, given a parameter $\alpha \geq 1$, lower bounds the norm $\|f\|_p$ by a $1/\alpha$-fraction of the L_p mass of the stream had all updates been positive. Here, for a vector f, $\|f\|_p = \left(\sum_{i=1}^n |f_i|^p\right)^{1/p}$, and the value of p we choose depends on the application. This gives a fluid medium between insertion only streams (with $\alpha = 1$), and turnstile streams (with $\alpha = \text{poly}(n)$), and allows for analysis in terms of α.

We show that for streams with this α-property, for many fundamental streaming problems we can replace a $O(\log(n))$ factor in the space usage for algorithms in the turnstile model with a $O(\log(\alpha))$ factor. This is true for identifying heavy hitters, inner product estimation, L_0 estimation, L_1 estimation, L_1 sampling, and support sampling. For each problem, we give matching or nearly matching lower bounds for α-property streams. We note that in practice, many important turnstile data streams are in fact α-property streams for small values of α. For such applications, our results represent significant improvements in efficiency for all the aforementioned problems.

ACM Reference Format:
Rajesh Jayaram and David P. Woodruff. 2018. Data Streams with Bounded Deletions: Extended Abstract. In *PODS'18: 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3196959.3196986

*See [35] for the full version of this work.

1 INTRODUCTION

Data streams have become increasingly important in modern applications, where the sheer size of a dataset imposes stringent restrictions on the resources available to an algorithm. Examples of such datasets include internet traffic logs, sensor networks, financial transaction data, database logs, and scientific data streams (such as huge experiments in particle physics, genomics, and astronomy). Given their prevalence, there is a substantial body of literature devoted to designing extremely efficient one-pass algorithms for important data stream problems. We refer the reader to [8, 48] for surveys of these algorithms and their applications.

Formally, a data stream is given by an underlying vector $f \in \mathbb{R}^n$, called the *frequency vector*, which is initialized to 0^n. The frequency vector then receives a stream of m updates of the form $(i_t, \Delta_t) \in [n] \times \{-M, \dots, M\}$ for some $M > 0$ and $t \in [m]$. The update (i, Δ) causes the change $f_{i_t} \leftarrow f_{i_t} + \Delta_t$. For simplicity, we make the common assumption that $\log(mM) = O(\log(n))$, though our results generalize naturally to arbitrary n, m [11].

Two well-studied models in the data stream literature are the *insertion-only* model and the *turnstile* model. In the former model, it is required that $\Delta_t > 0$ for all $t \in [m]$, whereas in the latter Δ_t can be any integer in $\{-M, \dots, M\}$. It is known that there are significant differences between these models. For instance, identifying an index $i \in [n]$ for which $|x_i| > \frac{1}{10}\sum_{j=1}^n |x_j|$ can be accomplished with only $O(\log(n))$ bits of space in the insertion-only model [9], but requires $\Omega(\log^2(n))$ bits in the turnstile model [37]. This $\log(n)$ gap between the complexity in the two models occurs in many other important streaming problems.

Due to this "complexity gap", it is perhaps surprising that no intermediate streaming model had been systematically studied in the literature before. For motivation on the usefulness of such a model, we note that nearly all of the lower bounds for turnstile streams involve inserting a large number of items before deleting nearly all of them [36–38, 40]. This behavior seems unlikely in practice, as the resulting norm $\|f\|_p$ becomes arbitrarily small regardless of the size of the stream. In this paper, we introduce a new model which avoids the lower bounds for turnstile streams by lower bounding the norm $\|f\|_p$. We remark that while similar notions of bounded deletion streams have been considered for their practical applicability [25] (see also [20], where a bound on the maximum number of edges that could be deleted in a graph stream was useful), to the best of our knowledge there is no comprehensive theoretical study of data stream algorithms in this setting.

Formally, we define the *insertion vector* $I \in \mathbb{R}^n$ to be the frequency vector of the substream of positive updates ($\Delta_t \geq 0$), and the *deletion vector* $D \in \mathbb{R}^n$ to be the entry-wise absolute value of

Problem	Turnstile L.B.	α-Property U.B.	Citation	Notes
ϵ-Heavy Hitters	$\Omega(\epsilon^{-1}\log^2(n))$	$O(\epsilon^{-1}\log(n)\log(\alpha))$	[37]	Strict-turnstile, succeeds w.h.p.
ϵ-Heavy Hitters	$\Omega(\epsilon^{-1}\log^2(n))$	$O(\epsilon^{-1}\log(n)\log(\alpha))$	[37]	General-turnstile, $\delta = O(1)$
Inner Product	$\Omega(\epsilon^{-1}\log(n))$	$O(\epsilon^{-1}\log(\alpha))$	Theorem 8.8	General-turnstile
L_1 Estimation	$\Omega(\log(n))$	$O(\log(\alpha))$	Theorem 8.4	Strict-turnstile
L_1 Estimation	$\Omega(\epsilon^{-2}\log(n))$	$O(\epsilon^{-2}\log(\alpha)) + \log(n))$	[38]	General-turnstile
L_0 Estimation	$\Omega(\epsilon^{-2}\log(n))$	$O(\epsilon^{-2}\log(\alpha)) + \log(n))$	[38]	General-turnstile
L_1 Sampling	$\Omega(\log^2(n))$	$O(\log(n)\log(\alpha))$	[37]	General-turnstile $(*)$
Support Sampling	$\Omega(\log^2(n))$	$O(\log(n)\log(\alpha))$	[40]	Strict-turnstile

Figure 1: The best known lower bounds (L.B.) for classic data stream problems in the turnstile model, along with the upper bounds (U.B.) for α-property streams from this paper. The notes specify whether an U.B./L.B. pair applies to the strict or general turnstile model. For simplicity, we have suppressed $\log\log(n)$ and $\log(1/\epsilon)$ terms, and all results are for $\delta = O(1)$ failure probability, unless otherwise stated. $(*)$ L_1 sampling note: strong α-property, with $\epsilon = \Theta(1)$ for both U.B. & L.B.

the frequency vector of the substream of negative updates. Then $f = I - D$ by definition. Our model is as follows.

Definition 1.1. For $\alpha \geq 1$ and $p \geq 0$, a data stream f satisfies the L_p *α-property* if $\|I + D\|_p \leq \alpha \|f\|_p$.

For $p = 1$, the definition simply asserts that the final L_1 norm of f must be no less than a $1/\alpha$ fraction of the total weight of updates in the stream $\sum_{t=1}^m |\Delta_t|$. For strict turnstile streams, this is equivalent to the number of deletions being less than a $(1 - 1/\alpha)$ fraction of the number of insertions, hence a bounded deletion stream.

For $p = 0$, the α-property simply states that $\|f\|_0$, the number of non-zero coordinates at the end of the stream, is no smaller than a $1/\alpha$ fraction of the number of distinct elements seen in the stream (known as the F_0 of the stream). Importantly, note that for both cases this constraint need only hold at the time of query, and not necessarily at every point in the stream.

Observe for $\alpha = 1$, we recover the insertion-only model, whereas for $\alpha = mM$ in the L_1 case or $\alpha = n$ in the L_0 case we recover the turnstile model (with the minor exception of streams with $\|f\|_p = 0$). So α-property streams are a natural parameterized intermediate model between the insertion-only and turnstile models. For clarity, we use the term *unbounded deletion stream* to refer to a (general or strict) turnstile stream which does not satisfy the α-property for $\alpha = o(n)$.

For many applications of turnstile data streaming algorithms, the streams in question are in fact α-property streams for small values of α. For instance, in network traffic monitoring it is useful to estimate differences between network traffic patterns across distinct time intervals or routers [48]. If f_i^1, f_i^2 represent the number of packets sent between the i-th [source, destination] IP address pair in the first and second intervals (or routers), then the stream in question is $f^1 - f^2$. In realistic systems, the traffic behavior will not be identical across days or routers, and even differences as small as 0.1% in overall traffic behavior (i.e. $\|f^1 - f^2\|_1 > .001\|f^1 + f^2\|_1$) will result in $\alpha < 1000$ (which is significantly smaller than the theoretical universe size of $n \approx 2^{256}$ potential IP addresses pairs in IPv6).

A similar case for small α can be made for differences between streams whenever these differences are not arbitrarily small. This includes applications in streams of financial transactions, sensor network data, and telecommunication call records [19, 33], as well as for identifying newly trending search terms, detecting DDoS attacks, and estimating the spread of malicious worms [23, 43, 46, 49, 55].

A setting in which α is also likely to be small is database analytics. For instance, an important tool for database synchronization is Remote Differential Compression (RDC)[3, 53], which allows similar files to be compared between a client and server by transmitting only the differences between them. For files given by large data streams, one can feed these differences back into sketches of the file to complete the synchronization. Even if as much as a half of the file must be resynchronized between client and sever (an inordinately large fraction for typical RDC applications), streaming algorithms with $\alpha = 2$ would suffice to recover the data.

For L_0, there are important applications of streams with bounded ratio $\alpha = F_0/L_0$. For example, L_0 estimation is applicable to networks of cheap moving sensors, such as those monitoring wildlife movement or water-flow patterns [33]. In such networks, some degree of clustering is expected (in regions of abundant food, points of water-flow accumulation), and these clusters will be consistently occupied by sensors, resulting in a bounded ratio of inactive to active regions. Furthermore, in networking one often needs to estimate the number of distinct IP addresses with active network connections at a given time [26, 48]. Here we also observe some degree of clustering on high-activity IP's, with persistently active IP's likely resulting in an L_0 to F_0 ratio much larger than $1/n$ (where n is the universe size of IP addresses).

In many of the above applications, it may be possible to treat α as a constant when compared with n. For such applications, the space improvements detailed in Figure 1 are considerable, and reduce the space complexity of the problems nearly or exactly to known upper bounds for insertion-only streams [9, 37, 39, 47].

Finally, we remark that in some cases it may not be unreasonable to assume that the magnitude of *every* coordinate would be bounded by some fraction of the updates to it. For instance, in the case of RDC it is seems possible that none of the files would be totally removed. We summarize this stronger guarantee as the *strong α-property.*

Definition 1.2. For $\alpha \geq 1$, a data stream f satisfies the *strong α-property* if $I_i + D_i \leq \alpha |f_i|$ for all $i \in [n]$.

Note that this property is strictly stronger that the L_p α-property for any $p \geq 0$. In particular, it forces $f_i \neq 0$ if i is updated in the stream. In this paper, however, we focus primarily on the more general α-property of Definition 1.1, and use α-property to refer to Definition 1.1 unless otherwise explicitly stated. Nevertheless, we show that our lower bounds for L_p heavy hitters, L_1 estimation, L_1 sampling, and inner product estimation, all hold even for the more restricted strong α-property streams.

A full version of this paper, along with all the omitted proofs, can be found at [35].

1.1 Our Contributions

We show that for many well-studied streaming problems, we can replace a $\log(n)$ factor in algorithms for general turnstile streams with a $\log(\alpha)$ factor for α-property streams. This is a significant improvement for small values of α. Our upper bound results, along with the lower bounds for the unbounded deletion case, are given in Figure 1. Several of our results come from the introduction of a new data structure, CSSampSim (Section 2), which produces point queries for the frequencies f_i with small additive error. Our improvements from CSSampSim and other L_1 problems are the result of new sampling techniques for α-property streams

While sampling of streams has been studied in many papers, most have been in the context of insertion only streams (see, e.g., [14, 16–18, 22, 30, 41, 44, 54]). Notable examples of the use of sampling in the presence of deletions in a stream are [15, 27–29, 32]. We note that these works are concerned with unbiased estimators and do not provide the $(1 \pm \epsilon)$-approximate relative error guarantees with small space that we obtain. They are also concerned with unbounded deletion streams, whereas our algorithms exploit the α-property of the underlying stream to obtain considerable savings.

In addition to upper bounds, we give matching or nearly matching lower bounds in the α-property setting for all the problems we consider. In particular, for the L_1-related problems (heavy hitters, L_1 estimation, L_1 sampling, and inner product estimation), we show that these lower bounds hold even for the stricter case of strong α-property streams.

We also demonstrate that for general turnstile streams, obtaining a constant approximation of the L_1 still requires $\Omega(\log(n))$-bits for α-property streams. For streams with unbounded deletions, there is an $\Omega(\epsilon^{-2} \log(n))$ lower bound for $(1 \pm \epsilon)$-approximation [38]. Although we cannot remove this $\log n$ factor for α-property streams, we are able to show an upper bound of $\tilde{O}(\epsilon^{-2} \log \alpha + \log n)$ bits of space for strong α-property streams, where the $\tilde{O}$ notation hides $\log(1/\epsilon) + \log \log n$ factors. We thus separate the dependence of ϵ^{-2} and $\log n$ in the space complexity, illustrating an additional benefit of α-property streams, and show a matching lower bound for strong α-property streams.

1.2 Our Techniques

Our results for L_1 streaming problems in Sections 2 to 5 are built off of the observation that for α-property streams the number of insertions and deletions made to a given $i \in [n]$ are both upper bounded by $\alpha \|f\|_1$. We can then think of sampling updates to i as a biased coin flip with bias at least $1/2 + (\alpha \|f\|_1)^{-1}$. By sampling $\text{poly}(\alpha/\epsilon)$ stream updates and scaling up, we can recover the difference between the number of insertions and deletions up to an additive $\epsilon \|f\|_1$ error.

To exploit this fact, in Section 2 we introduce a data structure CSSampSim inspired by the classic Countsketch of [13], which simulates running each row of Countsketch on a small uniform sample of stream updates. Our data structure does not correspond to a valid instantiation of Countsketch on any stream since we sample different stream updates for different rows of Countsketch. Nevertheless, we show via a Bernstein inequality that our data structure obtains the Countsketch guarantee plus an $\epsilon \|f\|_1$ additive error, with only a logarithmic dependence on ϵ in the space. This results in more efficient algorithms for the L_1 heavy hitters problem (Section 3), and is also used in our L_1 sampling algorithm (Section 4). We are able to argue that the counters used in our algorithms can be represented with much fewer than $\log n$ bits because we sample a very small number of stream updates.

Additionally, we demonstrate that sampling $\text{poly}(\alpha/\epsilon)$ updates preserves the inner product between α-property streams f, g up to an additive $\epsilon \|f\|_1 \|g\|_1$ error. Then by hashing the sampled universe down modulo a sufficiently large prime, we show that the inner product remains preserved, allowing us to estimate it in $O(\epsilon^{-1} \log(\alpha))$ space (Section 2.2).

Our algorithm for L_1 estimation (Section 5) utilizes our biased coin observation to show that sampling will recover the L_1 of a strict turnstile α-property stream. To carry out the sampling in $o(\log(n))$ space, give a alternate analysis of the well known Morris counting algorithm, giving better space but worse error bounds. This allows us to obtain a rough estimate of the position in the stream so that elements can be sampled with the correct probability. For L_1 estimation in general turnstile streams, we analyze a virtual stream which corresponds to scaling our input stream by Cauchy random variables, argue it still has the α-property, and apply our sampling analysis for L_1 estimation on it.

Our results for the L_0 streaming problems in Sections 6 and 7 mainly exploit the α-property in sub-sampling algorithms. Namely, many data structure for L_0 streaming problems subsample the universe $[n]$ at $\log(n)$ levels, corresponding to $\log(n)$ possible thresholds which could be $O(1)$-approximations of the L_0. If, however, an $O(1)$ approximation were known in advance, we could immediately subsample to this level and remove the $\log(n)$ factor from the space bound. For α-property streams, we note that the non-decreasing values F_0^t of F_0 after t updates must be bounded in the interval $[L_0^t, O(\alpha)L_0]$. Thus, by employing an $O(1)$ estimator R^t of F_0^t, we show that it suffices to subsample to only the $O(\log(\alpha/\epsilon))$ levels which are closest to $\log(R^t)$ at time t, from which our space improvements follows.

1.3 Preliminaries

If g is any function of the updates of the stream, for $t \in [m]$ we write g^t to denote the value of g after the updates $(i_1, \Delta_1), \ldots, (i_t, \Delta_t)$. For Sections 2 to 5, it will suffice to assume $\Delta_t \in \{-1, 1\}$ for all $t \in [m]$. For general updates, we can implicitly consider them to be several consecutive updates in $\{-1, 1\}$, and our analysis will hold in this expanded stream. This implicit expanding of updates is only necessary for our algorithms which sample updates with some probability p. If an update $|\Delta_t| > 1$ arrives, we update our data structures with the value $\text{sign}(\Delta_t) \cdot \text{Bin}(|\Delta_t|, p)$, where $\text{Bin}(n, p)$ is the binomial random variable on n trials with success probability p, which has the same effect. In this unit-update setting, the L_1 α-property reduces to $m \leq \alpha \|f^m\|_1$, so this is the definition which we will use for the L_1 α-property. We use the term *unbounded deletion stream* to refer to streams without the α-property (or equivalently streams that have the $\alpha = \text{poly}(n)$-property and can also have all 0 frequencies). For Sections 2 to 5, we will consider only the L_1 α-property, and thus drop the L_1 in these sections for simplicity, and for Sections 6 and 7 we will consider only the L_0 α-property, and similarly drop the L_0 there.

We call a vector $y \in \mathbb{R}^n$ *k-sparse* if it has at most k non-zero entries. For a given vector $f \in \mathbb{R}^n$, let $\text{Err}_p^k(f) = \min_{y\ k\text{-sparse}} \|f - y\|_p$. In other words, $\text{Err}_p^k(f)$ is the p-norm of f with the k heaviest entries removed. We call the argument minimizer of $\|f - y\|_p$ the best k-sparse approximation to f. Finally, we use the term *with high probability* (w.h.p.) to describe events that occur with probability $1 - n^{-c}$, where c is a constant. Events occur with low probability (w.l.p.) if they are a complement to a w.h.p. event. We will often ignore the precise constant c when it only factors into our memory usage as a constant.

2 FREQUENCY ESTIMATION VIA SAMPLING

In this section, we will develop many of the tools needed for answering approximate queries about α property streams. Primarily, we develop a data structure CSSampSim, inspired by the classic Countsketch of [13], that computes frequency estimates of items in the stream by sampling. This data structure will immediately result in an improved heavy hitters algorithm in Section 3, and is at the heart of our L_1 sampler in Section 4. In this section, we will write α-property to refer to the L_1 α-property throughout.

Firstly, for α-property streams, the following observation is crucial to many of our results. Given a fixed item $i \in [n]$, by sampling at least $\text{poly}(\alpha/\epsilon)$ stream updates we can preserve f_i (after scaling) up to an additive $\epsilon\|f\|_1$ error.

Lemma 2.1 (Sampling Lemma). *Let f be the frequency vector of a general turnstile stream with the α-property, and let f^* be the frequency vector of a uniformly sampled substream scaled up by $\frac{1}{p}$, where each update is sampled uniformly with probability $p > \alpha^2\epsilon^{-3}\log(\delta^{-1})/m$. Then with probability at least $1 - \delta$ for $i \in [n]$, we have*

$$|f_i^* - f_i| < \epsilon\|f\|_1$$

Moreover, we have $\sum_{i=1}^n f_i^* = \sum_{i=1}^n f_i \pm \epsilon\|f\|_1$.

Proof. Assume we sample each update to the stream independently with probability p. Let f_i^+, f_i^- be the number of insertions and deletions of element i respectively, so $f_i = f_i^+ - f_i^-$. Let X_j^+ indicate that the j-th insertion to item i is sampled. First, if $\epsilon\|f\|_1 < f_i^+$ then by Chernoff bounds:

$$\Pr[|\frac{1}{p}\sum_{j=1}^{f_i^+} X_j^+ - f_i^+| \geq \epsilon\|f\|_1] \leq 2\exp\big(\frac{-pf_i^+(\epsilon\|f\|_1)^2}{3(f_i^+)^2}\big)$$
$$\leq \exp\big(-p\epsilon^3 m/\alpha^2\big)$$

where the last inequality holds because $f_i^+ \leq m \leq \alpha\|f\|_1$. Taking $p \geq \alpha^2\log(1/\delta)/(\epsilon^3 m)$ gives the desired probability δ. Now if $\epsilon\|f\|_1 \geq f_i^+$, then $\Pr[\frac{1}{p}\sum_{j=1}^{f_i^+} X_j^+ \geq f_i^+ + \epsilon\|f\|_1] \leq \exp\big(-pf_i^+\epsilon\|f\|_1/f_i^+\big) \leq \exp\big(-p\epsilon m/\alpha\big) \leq \delta$ for the same value of p. Applying the same argument to f_i^-, we obtain $f_i^* = f_i^+ - f_i^- \pm 2\epsilon\|f\|_1 = f_i \pm 2\epsilon\|f\|_1$ as needed after rescaling ϵ. For the final statement, we can consider all updates to the stream as being made to a single element i, and then simply apply the same argument given above. □

2.1 Count-Sketch Sampling Simulator

We now describe the Countsketch algorithm of [13], which is a simple yet classic algorithm in the data stream literature. For a parameter k and $d = O(\log(n))$, it creates a $d \times 6k$ matrix A initialized to 0, and for every row $i \in [d]$ it selects hash functions $h_i : [n] \to [6k]$, and $g_i : [n] \to \{1, -1\}$ from 4-wise independent uniform hash families. On every update (i_t, Δ_t), it hashes this update into every row $i \in [d]$ by updating $a_{i,h_i(i_t)} \leftarrow a_{i,h_i(i_t)} + g_i(i_t)\Delta_t$. It then outputs y^* where $y_j^* = \text{median}\{g_i(j)a_{i,h_i(j)} \mid i \in [d]\}$. The guarantee of one row of the Countsketch is as follows.

Lemma 2.2. *Let $f \in \mathbb{R}^n$ be the frequency vector of any general turnstile stream hashed into a $d \times 6k$ Countsketch table. Then with probability at least $2/3$, for a given $j \in [n]$ and row $i \in [d]$ we have $|g_i(j)a_{i,h_i(j)} - f_j| < k^{-1/2}\text{Err}_2^k(f)$. It follows if $d = O(\log(n))$, then with high probability, for all $j \in [n]$ we have $|y_j^* - f_j| < k^{-1/2}\text{Err}_2^k(f)$, where y^* is the estimate of Countsketch. The space required is $O(k\log^2(n))$ bits.*

We now introduce a data structure which simulates running Countsketch on a uniform sample of the stream of size $\text{poly}(\alpha\log(n)/\epsilon)$. The full data structure is given in Figure 2. Note that for a fixed row, each update is chosen with probability at least $2^{-p} \geq S/(2m) = \Omega(\alpha^2 T^2\log(n)/(\epsilon^2 m))$. We will use this fact to apply Lemma 2.1 with $\epsilon' = (\epsilon/T)$ and $\delta = 1/\text{poly}(n)$. The parameter T will be $\text{poly}(\log(n)/\epsilon)$, and we introduce it as a new symbol purely for clarity.

Now the updates to each row in CSSS are sampled independently from the other rows, thus CSSS may not represent running Countsketch on a single valid sample. However, each row *independently* contains the result of running a row of Countsketch on a valid sample of the stream. Since the Countsketch guarantee holds with probability 2/3 for each row, and we simply take the median of $O(\log(n))$ rows to obtain high probability, it follows that the output of CSSS will still satisfy an additive error bound w.h.p. if each row also satisfies that error bound with probability 2/3.

By Lemma 2.1 with sensitivity parameter (ϵ/T), we know that we preserve the weight of all items f_i with $|f_i| \geq 1/T\|f\|_1$ up to a $(1 \pm \epsilon)$ factor w.h.p. after sampling and rescaling. For all smaller elements, however, we obtain error additive in $\epsilon\|f\|_1/T$. This gives rise to the natural division of the coordinates of f. Let *big* $\subset [n]$ be the set of i with $|f_i| \geq 1/T\|f\|_1$, and let *small* $\subset [n]$ be the complement. Let $\mathcal{E} \subset [n]$ be the top k heaviest coordinates in f, and let $s \in \mathbb{R}^n$ be a fixed sample vector of f after rescaling by p^{-1}. Since $\text{Err}_2^k(s) = \min_{\hat{y}\ k\text{-sparse}} \|s - \hat{y}\|_2$, it follows that $\text{Err}_2^k(s) \leq \|s_{big \setminus \mathcal{E}}\|_2 + \|s_{small}\|_2$. Furthermore, by Lemma 2.1 $\|s_{big\setminus\mathcal{E}}\|_2 \leq$

```
CSSampSim (CSSS)
Input: sensitivity parameters $k \geq 1,\ \epsilon \in (0,1)$.
  (1) Set $S = \Theta(\frac{\alpha^2}{\epsilon^2}T^2\log(n))$, where $T \geq 4/\epsilon^2 + \log(n)$.
  (2) Instantiate $d \times 6k$ count-sketch table $A$, for $d = O(\log(n))$. For each table entry $a_{ij} \in A$, store two values $a^+_{ij}$ and $a^-_{ij}$, both initialized to 0.
  (3) Select 4-wise independent hash functions $h_i : [n] \to [6k]$, $g_i : [n] \to \{1,-1\}$, for $i \in [d]$.
  (4) Set $p \leftarrow 0$, and start $\log(n)$ bit counter to store the position in the stream.
  (5) On Update $(i_t, \Delta_t)$:
      (a) if $t = 2^r\log(S) + 1$ for any $r \geq 1$, then for every entry $a_{ij} \in A$ set $a^+_{ij} \leftarrow$Bin$(a^+_{ij}, 1/2)$, $a^-_{ij} \leftarrow$Bin$(a^-_{ij}, 1/2)$, and $p \leftarrow p+1$
      (b) On Update $(i_t, \Delta_t)$: Sample $(i_t, \Delta_t)$ with probability $2^{-p}$. If sampled, then for $i \in [d]$
          (i) if $\Delta_t g_i(i_t) > 0$, set $a^+_{i,h_i(i_t)} \leftarrow a^+_{ih_i(i_t)} + \Delta_t g_i(i_t)$.
          (ii) else set $a^-_{i,h_i(i_t)} \leftarrow a^-_{ih_i(i_t)} + |\Delta_t g_i(i_t)|$
  (6) On Query for $f_j$: return $y^*_j = \text{median}\{2^p g_i(j) \cdot (a^+_{i,h_i(j)} - a^-_{i,h_i(j)}) \mid i \in [d]\}$.
```

Figure 2: Our data structure to simulate running Countsketch on a uniform sample of the stream.

$(1+\epsilon)\|f_{big\setminus\mathcal{E}}\|_2 \leq (1+\epsilon)\text{Err}^k_2(f)$. So it remains to upper bound $\|s_{small}\|_2^2$, which we do in the following technical lemma.

The intuition for the Lemma is that $\|f_{small}\|_2$ is maximized when all the L_1 weight is concentrated in T elements, thus $\|f_{small}\|_2 \leq (T(\|f\|_1/T)^2)^{1/2} = \|f\|_1/T^{1/2}$. By the α property, we know that the number of insertions made to the elements of *small* is bounded by $\alpha\|f\|_1$. Thus, computing the variance of $\|s_{small}\|_2^2$ and applying Bernstein's inequality, we obtain a similar upper bound for $\|s_{small}\|_2$.

Lemma 2.3. *If s is the rescaled frequency vector resulting from uniformly sampling with probability $p \geq S/(2m)$, where $S = \Omega(\alpha^2 T^2 \log(n))$ for $T = \Omega(\log(n))$, of a general turnstile stream f with the α property, then we have $\|s_{small}\|_2 \leq 2T^{-1/2}\|f\|_1$ with high probability.*

Applying the result of Lemma 2.3, along with the bound on $\text{Err}^k_2(s)$ from the previous paragraphs, we obtain the following corollary.

Corollary 2.4. *With high probability, if s is as in Lemma 2.3, then $Err^k_2(s) \leq (1+\epsilon)Err^k_2(f) + 2T^{-1/2}\|f\|_1$*

Now we analyze the error from CSSS. Observe that each row in CSSS contains the result of hashing a uniform sample into $6k$ buckets. Let $s^i \in \mathbb{R}^n$ be the frequency vector, *after scaling* by $1/p$, of the sample hashed into the i-th row of CSSS, and let $y^i \in \mathbb{R}^n$ be the estimate of s^i taken from the i-th row of CSSS. Let $\sigma(i) : n \to [O(\log(n))]$ be the row from which Countsketch returns its estimate for f_i, meaning $y^*_i = y^{\sigma(i)}_i$.

Theorem 2.5. *For $\epsilon > 0, k > 1$, with high probability, when run on a general turnstile stream $f \in \mathbb{R}^n$ with the α property, CSSS with $6k$ columns, $O(\log(n))$ rows, returns y^* such that, for every $i \in [n]$ we have*

$$|y^*_i - f_i| \leq 2(\frac{1}{k^{1/2}}Err^k_2(f) + \epsilon\|f\|_1)$$

It follows that if $\hat{y}$ is the best k-sparse approximation to y^*, then $Err^k_2(f) \leq \|f-\hat{y}\|_2 \leq 5(k^{1/2}\epsilon\|f\|_1 + Err^k_2(f))$ with high probability. The space required is $O(k\log(n)\log(\alpha\log(n)/\epsilon))$.

Proof. Set $S = \Theta(\frac{\alpha^2}{\epsilon^2}T^2\log(n))$ as in Figure 2, and set $T = 4/\epsilon^2 + O(\log(n))$. Fix any $i \in [n]$. Now CSSS samples updates uniformly with probability $p > S/(2m)$, so applying Lemma 2.1 to our sample s^j_i of f_i for each row j and union bounding, with high probability we have $s^j_i = f_i \pm \frac{\epsilon}{T}\|f\|_1 = s^q_i$ for all rows j, q. Then by the Countsketch guarantee of Lemma 2.2, for each row q that $y^q_i = f_i \pm (\frac{\epsilon}{T}\|f\|_1 + k^{-1/2}\max_j \text{Err}^k_2(s^j))$ with probability 2/3. Thus $y^*_i = y^{\sigma(i)}_i = f_i \pm (\frac{\epsilon}{T}\|f\|_1 + k^{-1/2}\max_j\text{Err}^k_2(s^j))$ with high probability. Now noting that $\sqrt{T} \geq 2/\epsilon$, we apply Corollary 2.4 to $\text{Err}^k_2(s^j)$ and union bound over all $j \in [O(\log(n))]$ to obtain $\max_j \text{Err}^k_2(s^j) \leq (1+\epsilon)\text{Err}^k_2(f) + \epsilon\|f\|_1$ w.h.p., and union bounding over all $i \in [n]$ gives $|y^*_i - f_i| \leq 2(\frac{1}{k^{1/2}}\text{Err}^k_2(f) + \epsilon\|f\|_1)$ for all $i \in [n]$ with high probability.

For the second claim, note that $\text{Err}^k_2(f) \leq \|f - f'\|_2$ for any k-sparse vector f', from which the first inequality follows. Now if the top k coordinates are the same in y^* as in f, then $\|f - \hat{y}\|_2$ is at most $\text{Err}^k_2(f)$ plus the L_2 error from estimating the top k elements, which is at most $(4k(\epsilon\|f\|_1 + k^{-1/2}\text{Err}^k_2(f))^2)^{1/2} \leq 2(k^{1/2}\epsilon\|f\|_1 + \text{Err}^k_2(f))$. In the worst case the top k coordinates of y^* are disjoint from the top k in f. Applying the triangle inequality, the error is at most the error on the top k coordinates of y^* plus the error on the top k in f. Thus $\|f - \hat{y}\|_2 \leq 5(k^{1/2}\epsilon\|f\|_1 + \text{Err}^k_2(f))$ as required.

For the space bound, note that the Countsketch table has $O(k\log(n))$ entries, each of which stores two counters which holds $O(S)$ samples in expectation. So the counters never exceed $\text{poly}(S) = \text{poly}(\frac{\alpha}{\epsilon}\log(n))$ w.h.p. by Chernoff bounds, and so can be stored using $O(\log(\alpha\log(n)/\epsilon))$ bits each (we can simply terminate if a counter gets too large). □

We now address how the error term of Theorem 2.5 can be estimated so as to bound the potential error. This will be necessary for our L_1 sampling algorithm.

Lemma 2.6. *For $k > 1, \epsilon \in (0,1)$, given a α-property stream f, there is an algorithm that can produce an estimate v such that $Err^k_2(f) \leq v \leq 45k^{1/2}\epsilon\|f\|_1 + 20Err^k_2(f)$ with high probability. The space required is the space needed to run two instances of CSSS with paramters k, ϵ.*

2.2 Inner Products

Given two streams $f, g \in \mathbb{R}^n$, the problem of estimating the inner product $\langle f, g\rangle = \sum_{i=1}^n f_i g_i$ has attracted considerable attention in the streaming community for its usage in estimating the size of join and self-join relations for databases [5, 50, 51]. For unbounded deletion streams, to obtain an $\epsilon\|f\|_1\|g\|_1$-additive error approximation the best known result requires $O(\epsilon^{-1}\log(n))$ bits with a constant probability of success [21]. We show in Theorem 8.8 that $\Omega(\epsilon^{-1}\log(\alpha))$ bits are required even when f, g are *strong* α-property streams. This also gives a matching $\Omega(\epsilon^{-1}\log(n))$ lower bound for the unbounded deletion case

In Theorem 2.7, we give a matching upper bound for α-property streams up to $\log\log(n)$ and $\log(1/\epsilon)$ terms.

Theorem 2.7. *Given two general-turnstile stream vectors f, g with the α-property, there is a one-pass algorithm which with probability* $11/13$ *produces an estimate* $\texttt{IP}(f, g)$ *such that* $\texttt{IP}(f, g) = \langle f, g\rangle \pm O(\epsilon)\|f\|_1\|g\|_1$ *using* $O(\epsilon^{-1}\log(\frac{\alpha\log(n)}{\epsilon}))$ *bits of space.*

3 L_1 HEAVY HITTERS

As an application of the Countsketch sampling algorithm presented in the last section, we give an improved upper bound for the classic L_1 ϵ-heavy hitters problem in the α-property setting. Formally, given $\epsilon \in (0, 1)$, the L_1 ϵ-heavy hitters problem asks to return a subset of $[n]$ that contains all items i such that $|f_i| \geq \epsilon\|f\|_1$, and no items j such that $|f_j| < (\epsilon/2)\|f\|_1$.

The heavy hitters problem is one of the most well-studied problems in the data stream literature. For general turnstile unbounded deletion streams, there is a known lower bound of $\Omega(\epsilon^{-1}\log(n)\log(\epsilon n))$ (see [7], in the language of compressed sensing, and [37]), and the Countsketch of [13] gives a matching upper bound (assuming $\epsilon^{-1} = o(n)$). In the insertion only case, however, the problem can be solved using $O(\epsilon^{-1}\log(n))$ bits [9], and for the strictly harder L_2 heavy hitters problem (where $\|f\|_1$ is replaced with $\|f\|_2$ in the problem definition), there is an $O(\epsilon^{-2}\log(1/\epsilon)\log(n))$-bit algorithm [10]. In this section, we beat the lower bounds for unbounded deletion streams in the α-property case. We first run a subroutine to obtain a value $R = (1 \pm 1/8)\|f\|_1$ with probability $1-\delta$. To do this, we use the following algorithm from [38].

Fact 1 ([38]). *There is an algorithm which gives a* $(1 \pm \epsilon)$ *multiplicative approximation with probability* $1-\delta$ *of the value* $\|f\|_1$ *using space* $O(\epsilon^{-2}\log(n)\log(1/\delta))$.

Next, we run an instance of CSSS with parameters $k = 32/\epsilon$ and $\epsilon/32$ to obtain our estimate y^* of f. This requires space $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon}))$, and by Theorem 2.5 gives an estimate $y^* \in \mathbb{R}^n$ such that $|y_i^* - f_i| < 2(\sqrt{\epsilon/32}\text{Err}_2^k(f) + \epsilon\|f\|_1/32)$ for all $i \in [n]$ with high probability. We then return all items i with $|y_i^*| \geq 3\epsilon R/4$. Since the top $1/\epsilon$ elements do not contribute to $\text{Err}_2^k(f)$, the quantity is maximized by having k elements with weight $\|f\|_1/k$, so $\text{Err}_2^k(f) \leq k^{-1/2}\|f\|_1$. Thus $\|y_i^* - f\|_\infty < (\epsilon/8)\|f\|_1$.

Given this, it follows that for any $i \in [n]$ if $|f_i| \geq \epsilon\|f\|_1$, then $|y_i^*| > (7\epsilon/8)\|f\|_1 > (3\epsilon/4)R$. Similarly if $|f_i| < (\epsilon/2)\|f\|_1$, then $|y_i^*| < (5\epsilon/8)\|f\|_1 < (3\epsilon/4)R$. So our algorithm correctly distinguishes ϵ heavy hitters from items with weight less than $\epsilon/2$. The probability of failure is $O(n^{-c})$ from CSSS and δ for estimating R, and the space required is $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon}))$ for running CSSS and $O(\log(n)\log(1/\delta))$ to obtain the estimate R. This gives the following theorem.

Theorem 3.1. *Given* $\epsilon \in (0, 1)$*, there is an algorithm that solves the* ϵ*-heavy hitters problem for general turnstile* α*-property streams with probability* $1-\delta$ *using space* $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon}) + \log(n)\log(1/\delta))$.

Now note for strict turnstile streams, we can compute $R = \|f\|_1$ exactly with probability 1 using an $O(\log(n))$-bit counter. Since the error bounds from CSSS holds with high probability, we obtain the following result.

Theorem 3.2. *Given* $\epsilon \in (0, 1)$*, there is an algorithm that solves the* ϵ*-heavy hitters problem for strict turnstile* α*-property streams with high probability using space* $O(\epsilon^{-1}\log(n)\log(\alpha\ \log(n)/\epsilon))$.

4 L_1 SAMPLING

Another problem of interest is the problem of designing L_p samplers. First introduced by Monemizadeh and Woodruff in [45], it has since been observed that L_p samplers lead to alternative algorithms for many important streaming problems, such as heavy hitters, L_p estimation, and finding duplicates [6, 37, 45].

Formally, given a data stream frequency vector f, the problem of returning an ϵ-approximate relative error uniform L_p sampler is to design an algorithm that returns an index $i \in [n]$ such that

$$\Pr[i = j] = (1 \pm \epsilon)\frac{|f_j|^p}{\|f\|_p^p}$$

for every $j \in [n]$. An approximate L_p sampler is allowed to fail with some probability δ, however in this case it must not output any index. For the case of $p = 1$, the best known upper bound is $O(\epsilon^{-1}\log(\epsilon^{-1})\log^2(n)\log(\delta^{-1}))$ bits of space, and there is also an $\Omega(\log^2(n))$ lower bound for L_p samplers with $\epsilon = O(1)$ for any p [37]. In this section, using the data structure CSSS of Section 2, we will design an L_1 sampler for strict-turnstile strong L_1 α-property streams using $O(\epsilon^{-1}\log(\epsilon^{-1})\log(n)\log(\frac{\alpha\log(n)}{\epsilon})\log(\delta^{-1}))$ bits of space. Throughout the section we use *α-property* to refer to the L_1 α-property.

4.1 The L_1 Sampler

Our algorithm employs the technique of *precision sampling* in a similar fashion as in the L_1 sampler of [37]. The idea is to scale every item f_i by $1/t_i$ where $t_i \in [0, 1]$ is a uniform random variable, and return any index i such that $z_i = |f_i|/t_i > \frac{1}{\epsilon}\|f\|_1$, since this occurs with probability exactly $\epsilon\frac{|f_i|}{\|f\|_1}$. One can then run a traditional Countsketch on the scaled stream z to determine when an element passes this threshold.

In this section, we will adapt this idea to *strong* α-property streams (Definition 1.2). The necessity of the strong α-property arises from the fact that if f has the strong α-property, then any coordinate-wise scaling z of f still has the α-property with probability 1. Thus the stream z given by $z_i = f_i/t_i$ has the α-property (in fact, it again has the strong α-property, but we will only need the fact that z has the α-property). Our full L_1 sampler is given in Figure 3.

```
αL1Sampler: $L_1$ sampling algorithm for strict-turnstile strong $\alpha$-property streams:
Initialization
  (1) Instantiate CSSS with $k = O(\log(\epsilon^{-1}))$ columns and parameter $\epsilon' = \epsilon^3/\log^2(n)$.
  (2) Select $k = O(\log(\frac{1}{\epsilon}))$-wise independent uniform scaling factors $t_i \in [0,1]$ for $i \in [n]$.
  (3) Run CSSS on scaled input $z$ where $z_i = f_i/t_i$.
  (4) Keep $\log(n)$-bit counters $r, q$ to store $r = \|f\|_1$ and $q = \|z\|_1$.
Recovery
  (1) Compute estimate $y^*$ via CSSS.
  (2) Via algorithm of Lemma 2.6, compute $v$ such that $\text{Err}_2^k(z) \leq v \leq 45\frac{k^{1/2}\epsilon^3}{\log^2(n)}\|z\|_1 + 20\text{Err}_2^k(z)$.
  (3) Find $i$ with $|y_i^*|$ maximal.
  (4) If $v > k^{1/2}r + 45\frac{k^{1/2}\epsilon^3}{\log^2(n)}q$, or $|y_i^*| < \max\{\frac{1}{\epsilon}r, \frac{(c/2)\epsilon^2}{\log^2(n)}q\}$ where the constant $c > 0$ is as in Proposition 1, output FAIL, otherwise output $i$
      and $t_i y_i^*$ as the estimate for $f_i$.
```

Figure 3: Our L_1 sampling algorithm with sucsess probability $\Theta(\epsilon)$

By running CSSS to find the heavy hitters of z, we introduce error additive in $O(\epsilon'\|z\|_1) = O(\epsilon^3/\log^2(n)\|z\|_1)$, but as we will see the heaviest item in z is an $\Omega(\epsilon^2/\log^2(n))$ heavy hitter with probability $1 - O(\epsilon)$ conditioned on an arbitrary value of t_i, so this error will only be an $O(\epsilon)$ fraction of the weight of the maximum weight element. Note that we use the term c-heavy hitter for $c \in (0,1)$ to denote an item with weight at least $c\|z\|_1$. Our algorithm then attempts to return an item z_i which crosses the threshold $\|f\|_1/\epsilon$, and we will be correct in doing so if the tail error $\text{Err}_2^k(z)$ from CSSS is not too great.

To determine if this is the case, since we are in the strict turnstile case we can compute $r = \|f\|_1$ and $q = \|z\|_1$ exactly by keeping a $\log(n)$-bit counter (note however that we will only need constant factor approximations for these). Next, using the result of Lemma 2.6 we can accurately estimate $\text{Err}_2^k(z)$, and abort if it is too large in Recovery Step 4 of Figure 3. If the conditions of this step hold, we will be guaranteed that if i is the maximal element, then $y_i^* = (1 \pm O(\epsilon))z_i$. This allows us to sample ϵ-approximately, as well as guarantee that our estimate of z_i has relative error ϵ. We now begin our analysis our L_1 sampler. First, the proof of the following fact can be found in [37].

Lemma 4.1. *Given that the values of t_i are $k = \log(1/\epsilon)$-wise independent, then conditioned on an arbitrary fixed value $t = t_l \in [0,1]$ for a single $l \in [n]$, we have $\Pr[20\text{Err}_2^k(z) > k^{1/2}\|f\|_1] = O(\epsilon + n^{-c})$.*

The following proposition shows that the $\epsilon/\log^2(n)$ term in the additive error of our CSSS will be an ϵ fraction of the maximal element with high probability.

Proposition 1. *There exists some constant $c > 0$ such that conditioned on an arbitrary fixed value $t = t_l \in [0,1]$ for a single $l \in [n]$, if j is such that $|z_j|$ is maximal, then with probability $1 - O(\epsilon)$ we have $|z_j| \geq c\epsilon^2/\log^2(n)\|z\|_1$.*

Lemma 4.2. *The probability that αL1Sampler outputs the index $i \in [n]$ is $(\epsilon \pm O(\epsilon^2))\frac{|f_i|}{\|f\|_1} + O(n^{-c})$. The relative error of the estimate of f_i is $O(\epsilon)$ with high probability.*

Theorem 4.3. *For $\epsilon, \delta > 0$, there is an $O(\epsilon)$-relative error one-pass L_1 sampler for α-property streams which also returns an $O(\epsilon)$-relative error approximation of the returned item. The algorithm outputs FAIL with probability at most δ, and the space is $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon})\log(n)\log(\alpha \log(n)/\epsilon)\log(\frac{1}{\delta}))$.*

Proof. By the last lemma, it follows that the prior algorithm fails with probability at most $1 - \epsilon + O(n^{-c})$. Conditioned on the fact that an index i is output, the probability that $i = j$ is $(1 \pm O(\epsilon))\frac{|f_i|}{\|f\|_1} + O(n^{-c})$. By running $O(1/\epsilon \log(1/\delta))$ copies of this algorithm in parallel and returning the first index returned by the copies, we obtain an $O(\epsilon)$ relative error sampler with failure probability at most δ. The $O(\epsilon)$ relative error estimation of f_i follows from Lemma 4.2.

For space, note that CSSS requires $O(k\log(n)\log(\alpha\log(n)/\epsilon) = O(\log(n)\log(1/\epsilon)\log(\frac{\alpha\log(n)}{\epsilon}))$ bits of space, which dominates the cost of storing r, q and the cost of computing v via Lemma 2.6, as well as the cost of storing the randomness to compute k-wise independent scaling factors t_i. Running $O(1/\epsilon\log(1/\delta))$ copies in parallel gives the stated space bound. □

5 L_1 ESTIMATION

We now consider the well-studied L_1 estimation problem in the α-property setting (in this section we write α-property to refer to the L_1 α-property). We remark that in the general turnstile unbounded deletion setting, an $O(1)$ estimation of $\|f\|_1$ can be accomplished in $O(\log(n))$ space [38]. We show in Section 8, however, that even for α as small as $3/2$, estimating $\|f\|_1$ in general turnstile α-property streams still requires $\Omega(\log(n))$-bits of space. Nevertheless, in 5.2 we show that for α-property general turnstile streams there is a $\tilde{O}(\epsilon^{-2}\log(\alpha) + \log(n))$ bits of space algorithm, where $\tilde{O}$ hides $\log(1/\epsilon)$ and $\log\log(n)$ terms, thereby separating the ϵ^{-2} and $\log n$ factors. Furthermore, we show a nearly matching lower bound of $\Omega(\frac{1}{\epsilon^2}\log(\epsilon^2\alpha))$ for the problem (Theorem 8.3).

5.1 Strict Turnstile L_1 Estimation

Now for strict-turnstile α-property streams, we show that the problem can be solved with $\tilde{O}(\log(\alpha))$-bits. Ideally, to do so we would sample $\text{poly}(\alpha\log(n)/\epsilon)$ updates uniformly from the stream and apply Lemma 2.1. To do this without knowing the length of the stream in advance, we sample in exponentially increasing intervals, throwing away a prefix of the stream. At any given time, we will

```
$\alpha L_1$Estimator: Input $(\epsilon, \delta)$ to estimate $L_1$ of an $\alpha$-property strict turnstile stream.
(1) Initialization: Set $s \leftarrow O(\alpha^2\delta^{-1}\log^3(n)/\epsilon^2)$, and initialize Morris-Counter $v_t$ with parameter $\delta'$. Define $I_j = [s^j, s^{j+2}]$.
(2) Processing: on update $u_t$, for each $j$ such that $v_t \in I_j$, sample $u_t$ with probability $s^{-j}$.
(3) For each update $u_t$ sampled while $v_t \in I_j$, keep counters $c_j^+, c_j^-$ initialized to 0. Store all positive updates sampled in $c_j^+$, and (the absolute value of) all negative updates sampled in $c_j^-$.
(4) if $v_t \notin I_j$ for any $j$, delete the counters $c_j^+, c_j^-$.
(5) Return: $s^{-j^*}(c_{j^*}^+ - c_{j^*}^-)$ for which $j^*$ is such that $c_{j^*}^+, c_{j^*}^-$ have existed the longest (the stored counters which have been receiving updates for the most time steps).
```

Figure 4: L_1 Estimator for strict turnstile α-property streams.

sample at two different rates in two overlapping intervals, and we will return the estimate given by the sample corresponding to the interval from which we have sampled from the longest upon termination. We first give a looser analysis of the well known Morris counting algorithm.

LEMMA 5.1. *There is an algorithm,* Morris-Counter*, that given $\delta \in (0, 1)$ and a sequence of m events, produces non-decreasing estimates v_t of t such that*

$$\delta/(12\log(m))t \leq v_t \leq 1/\delta t$$

for a fixed $t \in [m]$ with probability $1 - \delta$. The algorithm uses $O(\log\log(m))$ bits of space.

Our full L_1 estimation algorithm is given in Figure 4. Note that the value s^{-j^*} can be returned symbolically by storing s and j^*, without explicitly computing the entire value. Also observe that we can assume that s is a power of 2 by rescaling, and sample with probability s^{-i} by flipping $\log(s)i$ fair coins sequentially and sampling only if all are heads, which requires $O(\log\log(n))$ bits of space.

THEOREM 5.2. *The algorithm αL_1Estimator gives a $(1 \pm \epsilon)$ approximation of the value $\|f\|_1$ of a strict turnstile stream with the α-property with probability $1 - \delta$ using $O(\log(\alpha/\epsilon) + \log(1/\delta) + \log(\log(n))))$ bits of space.*

PROOF. Let $\psi = 12\log^2(m)/\delta$. By the union bound on Lemma 5.1, with probability $1 - \delta$ the Morris counter v_t will produce estimates v_t such that $t/\psi \leq v_t \leq \psi t$ for all points $t = s^i/\psi$ and $t = \psi s^i$ for $i = 1, \dots, \log(m)/\log(s)$. Conditioned on this, I_j will be initialized by time ψs^j and not deleted before s^{j+2}/ψ for every $j = 1, 2, \dots, \log(n)/\log(s)$. Thus, upon termination, the oldest set of counters $c_{j^*}^+, c_{j^*}^-$ must have been receiving samples from an interval of size at least $m - 2\psi m/s$, with sampling probability $s^{-j^*} \geq s/(2\psi m)$. Since $s \geq 2\psi\alpha^2/\epsilon^2$, it follows by Lemma 2.1 that $s^{-j^*}(c_{j^*}^+ - c_{j^*}^-) = \sum_{i=1}^n \hat{f}_i \pm \epsilon\|\hat{f}\|_1$ w.h.p., where $\hat{f}$ is the frequency vector of all updates after time t^* and t^* is the time step where $c_{j^*}^+, c_{j^*}^-$ started receiving updates. By correctness of our Morris counter, we know that $t^* < 2\psi m/s < \epsilon\|f\|_1$, where the last inequality follows from the size of s and the the α-property, so the number of updates we missed before initializing $c_{j^*}^+, c_{j^*}^-$ is at most $\epsilon\|f\|_1$. Since f is a strict turnstile stream, $\sum_{i=1}^n \hat{f}_i = \|f\|_1 \pm t^* = (1 \pm O(\epsilon))\|f\|_1$ and $\|\hat{f}\|_1 = (1 \pm O(\epsilon))\|f\|_1$. After rescaling of ϵ we obtain $s^{-j^*}(c_{j^*}^+ - c_{j^*}^-) = (1 \pm \epsilon)\|f\|_1$ as needed.

For space, conditioned on the success of the Morris counter, which requires $O(\log\log(n))$-bits of space, we never sample from an interval I_j for more than ψs^{j+2} steps, and thus the maximum expected number of samples is ψs^2, and is at most s^3 with high probability by Chernoff bounds. Union bounding over all intervals, we never have more than s^2 samples in any interval with high probability. At any given time we store counters for at most 2 intervals, so the space required is $O(\log(s) + \log\log(m)) = O(\log(\alpha/\epsilon) + \log(1/\delta) + \log\log(n))$ as stated. □

Remark 1. *Note that if an update Δ_t to some coordinate i_t arrives with $|\Delta_t| > 1$, our algorithm must implicitly expand Δ_t to updates in $\{-1, 1\}$ by updating the counters by Sign(Δ_t)·Bin($|\Delta_t|, s^{-j}$) for some j. Note that computing this requires $O(\log(|\Delta_t|))$ bits of working memory, which is potentially larger than $O(\log(\alpha\log(n)/\epsilon))$. However, if the updates are streamed to the algorithm using $O(\log(|\Delta_t|))$ bits then it is reasonable to allow the algorithm to have at least this much working memory. Once computed, this working memory is no longer needed and does not factor into the space complexity of maintaining the sketch of αL_1*Estimator.

5.2 General Turnstile L_1 Estimator

In [38], an $O(\epsilon^{-2}\log(n))$-bit algorithm is given for general turnstile L_1 estimation. We show how modifications to this algorithm can result in improved algorithms for α-property streams. We state their algorithm in Figure 5, along with the results given in [38]. Here $\mathcal{D}_1$ is the distribution of a 1-stable random variable. In [34, 38], the variables $X = \tan(\theta)$ are used, where θ is drawn uniformly from $[-\frac{\pi}{2}, \frac{\pi}{2}]$. We refer the reader to [34] for a further discussion of p-stable distributions.

LEMMA 5.3 (A.6 [38]). *The entries of A, A' can be generated to precision $\delta = \Theta(\epsilon/m)$ using $O(k\log(n/\epsilon))$ bits.*

THEOREM 5.4 (THEOREM 2.2 [38]). *The algorithm above can be implemented using precision δ in the variables $A_{i,j}, A'_{i,j}$, and thus precision δ in the entries y_i, y'_i, such that the output $\tilde{L}$ satisfies $\tilde{L} = (1 \pm \epsilon)\|f\|_1$ with probability $3/4$, where $\delta = \Theta(\epsilon/m)$. In this setting, we have $y'_{med} = \Theta(1)\|f\|_1$, and*

$$\left|\left(\frac{1}{r}\sum_{i=1}^{r}\cos\left(\frac{y_i}{y'_{med}}\right)\right) - e^{-\left(\frac{\|f\|_1}{y'_{med}}\right)}\right| \leq O(\epsilon)$$

.

(1) **Initialization:** Generate random matrices $A \in \mathbb{R}^{r \times n}$ and $A \in \mathbb{R}^{r' \times n}$ of variables drawn from $\mathcal{D}_1$, where $r = \Theta(1/\epsilon^2)$ and $r' = \Theta(1)$. The variables A_{ij} are k-wise independent, for $k = \Theta(\log(1/\epsilon)/\log\log(1/\epsilon))$, and the variables A'_{ij} are k'-wise independent for $k' = \Theta(1)$. For $i \neq i'$, the seeds used to generate the variables $\{A_{i,j}\}_{j=1}^n$ and $\{A_{i',j}\}_{j=1}^n$ are pairwise independent
(2) **Processing:** Maintain vectors $y = Af$ and $y' = A'f$.
(3) **Return:** Let $y'_{med} = \text{median}\{|y'_i|\}_{i=1}^{r'}$. Output $\tilde{L} = y'_{med}\big(-\ln\big(\frac{1}{r}\sum_{i=1}^r \cos(\frac{y_i}{y'_{med}})\big)\big)$

Figure 5: L_1 estimator of [38] for general turnstile unbounded deletion streams.

We demonstrate that this algorithm can be implemented with reduced space complexity for α-property streams by sampling to estimate the values y_i, y'_i.

Theorem 5.5. *There is an algorithm that, given a general turnstile α-property stream f, produces an estimate $\tilde{L} = (1 \pm O(\epsilon))\|f\|_1$ with probability $2/3$ using $O(\epsilon^{-2}\log(\alpha\log(n)/\epsilon) + \frac{\log(\frac{1}{\epsilon})\log(n)}{\log\log(\frac{1}{\epsilon})})$ bits of space.*

6 L_0 ESTIMATION

The problem of estimating the support size of a stream is known as L_0 estimation. In other words, this is $L_0 = |\{i \in [n] \mid f_i \neq 0\}|$. L_0 estimation is a fundamental problem for network traffic monitoring, query optimization, and database analytics [1, 24, 52]. The problem also has applications in detecting DDoS attacks [4] and port scans [23].

For general turnstile streams, Kane, Nelson, and Woodruff gave an $O(\epsilon^{-2}\log(n)(\log(\epsilon^{-1}) + \log\log(n)))$-bit algorithm with constant probability of success [39], which nearly matches the known lower bound of $\Omega(\epsilon^{-2}\log(\epsilon^2 n))$ [38]. For insertion only streams, they also demonstrated an $O(\epsilon^{-2} + \log(n))$ upper bound. In this section we show that the ideas of [39] can be adapted to yield more efficient algorithms for general turnstile L_0 α-property streams. For the rest of the section, we will simply write *α-property* to refer to the L_0 α-property.

The idea of the algorithm stems from the observation that if $A = \Theta(K)$, then the number of non-empty bins after hashing A balls into K bins is well concentrated around its expectation. Treating this expectation as a function of A and inverting it, one can then recover A with good probability. By treating the (non-zero) elements of the stream as balls, we can hash the universe down into $K = 1/\epsilon^2$ bins and recover L_0 if $L_0 = \Theta(K)$. The primary challenge will be to ensure this last condition. In order to do so, we subsample the elements of the stream at $\log(n)$ levels, and simultaneously run an $O(1)$ estimator R of the L_0. To recover a $(1 \pm \epsilon)$ approximation, we use R to index into the level of subsampling corresponding to a substream with $\Theta(K)$ non-zero elements. We then invert the number of non-empty bins and scale up by a factor to account for the degree of subsampling.

6.1 Review of Unbounded Deletion Case

For sets U, V and integer k, let $\mathcal{H}_k(U,V)$ denote some k-wise independent hash family of functions mapping U into V. Assuming that $|U|, |V|$ are powers of 2, such hash functions can be represented using $O(k\log(|U| + |V|))$ bits [12] (without loss of generality we assume n, ϵ^{-1} are powers of 2 for the remainder of the section). For $x \in \mathbb{Z}_{\geq 0}$, we write $\text{lsb}(x)$ to denote the (0-based index of) the least significant bit of x written in binary. For instance, $\text{lsb}(6) = 1$ and $\text{lsb}(5) = 0$. We set $\text{lsb}(0) = \log(n)$. In order to fulfill the algorithmic template outlined above, we need to obtain a constant factor approximation R to L_0. This is done using the following result which can be found in [39].

Lemma 6.1. *Given a fixed constant $\delta > 0$, there is an algorithm,* `RoughL0Estimator`*, that with probability $1 - \delta$ outputs a value $R = \tilde{L}_0$ satisfying $L_0 \leq R \leq 110L_0$, using space $O(\log(n)\log\log(n))$.*

The main algorithm then subsamples the stream at $\log(n)$ levels. This is accomplished by choosing a hash function $h_1 : [n] \to \{0, \ldots, n-1\}$, and subsampling an item i at level $\text{lsb}(h_1(i))$. Then at each level of subsampling, the updates to the subsampled items are hashed into $K = \frac{1}{\epsilon^2}$ bins $k = \Omega(\log(\epsilon^{-1}/\log(\log(\epsilon^{-1}))))$-wise independently. The entire data structure is then represented by a $\log(n) \times K$ matrix B. The matrix B is stored modulo a sufficiently large prime p, and the updates to the rows are scaled via a random linear function to reduce the probability that deletions to one item cancel with insertions to another, resulting in false negatives in the number of buckets hit by items from the support. At the termination of the algorithm, we count the number T of non-empty bins in the i^*-th row of B, where $i^* = \max\{0, \log(\frac{16R}{K})\}$. We then return the value $\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1-T/K)}{\ln(1-1/K)}$. The full algorithm is given in Figure 6. First, the following Lemma can be found in [39].

Lemma 6.2. *There exists a constant ϵ_0 such that the following holds. Let A balls be mapped into $K = 1/\epsilon^2$ bins using a random $h \in \mathcal{H}_k([A],[K])$ for $k = c\log(1/\epsilon)/\log\log(1/\epsilon)$ for a sufficiently large constant c. Let X be a random variable which counts the number of non-empty bins after all balls are mapped, and assume $100 \leq A \leq K/20$ and $\epsilon \leq \epsilon_0$. Then $\mathbb{E}[X] = K(1 - (1 - K^{-1})^A)$ and*

$$\Pr[|X - \mathbb{E}[X]| \leq 8\epsilon\mathbb{E}[X]] \geq 4/5$$

Let A be the $\log(n) \times K$ bit matrix such that $A_{i,j} = 1$ iff there is at least one $v \in [n]$ with $f_v \neq 0$ such that $\text{lsb}(h_1(v)) = i$ and $h_3(h_2(v)) = j$. In other words, $A_{i,j}$ is an indicator bit which is 1 if an element from the support of f is hashed to the entry $B_{i,j}$ in the above algorithm. Clearly if $B_{i,j} \neq 0$, then $A_{i,j} \neq 0$. However, the other direction may not always hold. The proofs of the following lemma and theorem can be found in [39].

Lemma 6.3 (Lemma 6 of [39]). *Assuming that $L_0 \geq K/32$, with probability $3/4$, for all $j \in [K]$ we have $A_{i^*,j} = 0$ if and only if $B_{i^*,j} = 0$. Moreover, the space required to store each $B_{i,j}$ is $O(\log\log(n) + \log(1/\epsilon))$.*

```
L0Estimator: $L_0$ estimation algorithm
Input: $\epsilon > 0$
  (1) Set $K = 1/\epsilon^2$, and instantiate $\log(n) \times K$ matrix $B$ to 0.
  (2) Fix random $h_1 \in \mathcal{H}_2([n], \{0, \ldots, n-1\})$, $h_2 \in \mathcal{H}_2([n], [K^3])$, $h_3 \in \mathcal{H}_k([K^3], [K])$, and $h_4 \in \mathcal{H}_2([K^3], [K])$, for $k = \Omega(\log(\epsilon^{-1})/\log\log(\epsilon^{-1}))$.
  (3) Randomly choose prime $p \in [D, D^3]$, for $D = 100K\log(mM)$, and vector $\vec{u} \in \mathbb{F}_p^K$.
  (4) On Update $(i, \Delta)$: set
$B_{\text{lsb}(h_1(i)), h_3(h_2(i))} \longleftarrow \left(B_{\text{lsb}(h_1(i)), h_3(h_2(i))} + \Delta \cdot \vec{u}_{h_4(h_2(i))}\right) \pmod p$
  (5) Return: run RoughL0Estimator to obtain $R \in [L_0, 110L_0]$. Set $T = |\{j \in [K] \mid B_{i^*,j} \neq 0\}|$, where $i^* = \max\{0, \log(16R/K)\}$. Return estimate $\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1-T/K)}{\ln(1-1/K)}$.
```

Figure 6: L_0 Estimation Algorithm of [39]

```
$\alpha$L0Estimator: $L_0$ estimator for $\alpha$-property streams.
  (1) Initialize instance of L0Estimator, constructing only the top $2\log(4\alpha/\epsilon)$ rows of $B$. Let all parameters and hash functions be as in L0Estimator.
  (2) Initialize $\alpha$StreamRoughL0Est to obtain a value $\tilde{L}_0^t \in [L_0^t, 8\alpha L_0]$ for all $t \in [m]$, and set $\overline{L_0^t} = \max\{\tilde{L}_0^t, 8\log(n)/\log\log(n)\}$
  (3) Update the matrix $B$ as in Figure 6, but only store the rows with index $i$ such that $i = \log(16\overline{L_0^t}/K) \pm 2\log(4\alpha/\epsilon)$.
  (4) Return: run $\alpha$StreamConstL0Est to obtain $R \in [L_0, 100L_0]$, and set $T = |\{j \in [K] \mid B_{i^*,j} \neq 0\}|$, where $i^* = \log(16R/K)$. Return estimate $\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1-T/K)}{\ln(1-1/K)}$.
```

Figure 7: Our L_0 estimation algorithm for α-property streams with $L_0 > K/32$

Theorem 6.4. *Assuming that $L_0 > K/32$, the value returned by* L0Estimator *is a $(1 \pm \epsilon)$ approximation of the L_0 using space $O(\epsilon^{-2}\log(n)(\log(\frac{1}{\epsilon}) + \log(\log(n))\log(\frac{1}{\delta}))$, with 3/4 success probability.*

6.2 Dealing With Small L_0

In the prior section it was assumed that $L_0 \geq K/32 = \epsilon^{-2}/32$. We handle the estimation when this is not the case the same way as [39].

Lemma 6.5. *Let $\epsilon > 0$ be given and let $\delta > 0$ be a fixed constant. Then there is a subroutine using $O(\epsilon^{-2}(\log(\epsilon^{-1}) + \log\log(n)) + \log(n))$ bits of space which with probability $1 - \delta$ either returns a $(1 \pm \epsilon)$ approximation to L_0, or returns LARGE, with the guarantee that $L_0 > \epsilon^{-2}/16$.*

6.3 The Algorithm for α-Property Streams

We will give a modified version of the algorithm in Figure 6 for L_0 α property streams. Our algorithm is given in Figure 7. We note first that the return value of the unbounded deletion algorithm only depends on the row $i^* = \log(16R/K)$, and so we need only ensure that this row is stored. Our L_0 α-property implies that if L_0^t is the L_0 value at time t, then we must have $L_0^m = L_0 \geq 1/\alpha L_0^t$. So if we can obtain an $O(\alpha)$ approximation R^t to L_0^t, then at time t we need only maintain and sample the rows of the matrix with index within $c\log(\alpha/\epsilon)$ distance of $i^t = \log(16R^t/K)$, for some small constant c.

By doing this, the output of our algorithm will then be the same as the output of L0Estimator when run on the suffix of the stream beginning at the time when we first begin sampling to the row i^*. Since we begin sampling to this row when the current L_0^t is less than an ϵ fraction of the final L_0, it will follow that the L_0 of this suffix will be an ϵ-relative error approximation of the L_0 of the entire stream. Thus by the correctness of L0Estimator, the output of our algorithm will be a $(1 \pm \epsilon)^2$ approximation.

To obtain an $O(\alpha)$ approximation to L_0^t at all points t in the stream, we employ another algorithm of [39], which gives an $O(1)$ estimation of the F_0 value at all points in the stream, where $F_0 = |\{i \in [n] \mid f_i^t \neq 0 \text{ for some } t \in [m]\}|$. So by definition for any time $t \in [m]$ we have $F_0^t \leq F_0 = \|I + D\|_0 \leq \alpha\|f\|_0 = \alpha L_0$ by the α-property, and also by definition $F_0^t \geq L_0^t$ at all times t. These two facts together imply that $[F_0^t, 8F_0^t] \subseteq [L_0^t, 8\alpha L_0]$.

Lemma 6.6 ([39]). *There is an algorithm,* RoughF0Est*, that with probability $1-\delta$ outputs non decreasing estimates $\tilde{F_0}^t$ such that $\tilde{F_0}^t \in [F_0^t, 8F_0^t]$ for all $t \in [m]$ such that $F_0^t \geq \max\{8, \log(n)/\log\log(n)\}$, where m is the length of the stream. The spacedrequired is $O(\log(n)\log(\frac{1}{\delta}))$-bits.*

Corollary 6.7. *There is an algorithm, α*StreamRoughL0Est*, that with probability $1-\delta$ on an α-deletion stream outputs non-decreasing estimates $\tilde{L_0}^t$ such that $\tilde{L_0}^t \in [L_0^t, 8\alpha L_0]$ for all $t \in [m]$ such that $F_0^t \geq \max\{8, \log(n)/\log\log(n)\}$, where m is the length of the stream. The space required is $O(\log(n)\log(\frac{1}{\delta}))$ bits.*

Note that the approximation is only promised for t such that $F_0^t \geq \max\{8, \log(n)/\log\log(n)\}$. To handle this, we give a subroutine which produces the L_0 exactly for $F_0 < 8\log(n)/\log\log(n)$ using $O(\log(n))$ bits. Our main algorithm will assume that $F_0 > 8\log(n)/\log\log(n)$, and initialize its estimate of L_0^0 to be $\overline{L_0^0} = 8\log(n)/\log\log(n)$, where $\tilde{L}_0^t \in [L_0^t.8\alpha L_0]$ is the estimate produced by αStreamRoughL0Est.

```
α-SupportSampler: support sampling algorithm for α property streams.
Initialization:
    (1) Set $s \leftarrow 205k$, and initialize linear sketch function $J : \mathbb{R}^n \to \mathbb{R}^q$ for $q = O(s)$ via Lemma 7.1.
    (2) Select random $h \in \mathcal{H}_2([n],[n])$, set $I_j = \{i \in [n] \mid h(i) \le 2^j\}$, and set $\epsilon = 1/48$.
Processing:
    (1) Run αStreamRoughL0Est of Corollary 6.7 with $\delta = 1/12$ to obtain non-decreasing $R^t \in [L_0^t, 8\alpha L_0]$.
    (2) Let $B_t = \{j \in [\log(n)] \mid j = \log(ns/3R^t) \pm 2\log(\alpha/\epsilon)$ or $j \ge \log(ns\log\log(n)/(24\log(n)))\}$.
    (3) Let $t_j \in [m]$ be the first time $t$ such that $j \in B_t$ (if one exists). Let $t_j'$ be the first time step $t' > t_j$ such that $j \notin B_{t'}$ (if one exists).
    (4) For all $j \in B_t$, maintain linear sketch $x_j = J(f^{t_j:t_j'}|_{I_j})$.
Recovery:
    (1) For $j \in B_m$ at the end of the stream, attempt to invert $x_j$ into $f^{t_j:m}|_{I_j}$ via Lemma 7.1. Return all strictly positive coordinates of all successfully returned $f^{t_j:m}|_{I_j}$'s.
```

Figure 8: Our support sampling algorithm for α-property streams.

Lemma 6.8. *Given $c \ge 1$, there is an algorithm that with probability $49/50$ returns the L_0 exactly if $F_0 \le c$, and returns LARGE if $F_0 > c$. The space required is $O(c\log(c) + c\log\log(n) + \log(n))$ bits*

Finally, to remove the $\log(n)\log\log(n)$ memory overhead of running the RoughL0Estimator procedure to determine the row i^*, we show that the exact same $O(1)$ approximation of the final L_0 can be obtained using $O(\log(\alpha\log(n))\log(\log(n)) + \log(n))$ bits of space for α-property streams.

Lemma 6.9. *Given a fixed constant δ, there is an algorithm, αStreamConstL0Est that with probability $1-\delta$ when run on an α property stream outputs a value $R = \hat{L}_0$ satisfying $L_0 \le R \le 100L_0$, using space $O(\log(\alpha)\log\log(n) + \log(n))$.*

Theorem 6.10. *There is an algorithm that gives a $(1 \pm \epsilon)$ approximation of the L_0 value of a general turnstile stream with the α-property, using space $O(\frac{1}{\epsilon^2}\log(\frac{\alpha}{\epsilon})(\log(\frac{1}{\epsilon}) + \log\log(n)) + \log(n))$, with $2/3$ success probability.*

7 SUPPORT SAMPLING

The problem of support sampling asks, given a stream vector $f \in \mathbb{R}^n$ and a parameter $k \ge 1$, return a set $U \subset [n]$ of size at least $\min\{k, \|f\|_0\}$ such that for every $i \in U$ we have $f_i \neq 0$. Support samplers are needed crucially as subroutines for many dynamic graph streaming algorithms, such as connectivity, bipartitness, minimum spanning trees, min-cut, cut sparsifiers, spanners, and spectral sparsifiers [2]. They have also been applied to solve maximum matching [42], as well as hyperedge connectivity [31]. A more comprehensive study of their usefulness in dynamic graph applications can be found in [40].

For strict turnstile streams, an $\Omega(k\log^2(n/k))$ lower bound is known [40], and for general turnstile streams there is an $O(k\log^2(n))$ algorithm [37]. In this section we demonstrate that for L_0 α-property streams in the strict-turnstile case, more efficient support samplers exist. For the rest of the section, we write *α-property* to refer to the L_0 α-property, and we use the notation defined at the beginning of Section 6.1.

First consider the following template for the unbounded deletion case (as in [37]). First, we subsample the set of items $[n]$ at $\log(n)$ levels, where at level j, the set $I_j \subseteq [n]$ is subsampled with expected size $|I_j| = 2^j$. Let $f|_{I_j}$ be the vector f restricted to the coordinates of I_j (and 0 elsewhere). Then for each I_j, the algorithm creates a small sketch x_j of the vector $f|_{I_j}$. If $f|_{I_j}$ is sparse, we can use techniques from sparse recovery to recover $f|_{I_j}$ and report all the non-zero coordinates. We first state the following well known result which we utilize for this recovery.

Lemma 7.1 ([37]). *Given $1 \le s \le n$, there is a linear sketch and a recovery algorithm which, given $f \in \mathbb{R}^n$, constructs a linear sketch $J(f) : \mathbb{R}^n \to \mathbb{R}^q$ for $q = O(s)$ such that if f is s-sparse then the recovery algorithm returns f on input $J(f)$, otherwise it returns DENSE with high probability. The space required is $O(s\log(n))$ bits.*

Next, observe that for L_0 α-deletion streams, the value F_0^t is at least L_0^t and at most αL_0 for every $t \in [m]$. Therefore, if we are given an estimate of F_0^t, we show it will suffice to only subsample at $O(\log(\alpha))$-levels at a time. In order to estimate F_0^t we utilize the estimator αStreamRoughL0Est from Corollary 6.7 of Section 6. For $t' \ge t$, let $f^{t:t'} \in \mathbb{R}^n$ be the frequency vector of the stream of updates t to t'. We use the notation given in our full algorithm in Figure 8. Notice that since R^t is non-decreasing, once j is removed from B_t at time t_j' it will never enter again. So at the time of termination, we have $x_j = J(f^{t_j:m}|_{I_j})$ for all $j \in B_m$.

Theorem 7.2. *Given a strict turnstile stream f with the L_0 α-property and $k \ge 1$, the algorithm α-SupportSampler outputs a set $U \subset [n]$ such that $f_i \neq 0$ for every $i \in U$, and such that with probability $1-\delta$ we have $|U| \ge \min\{k, \|f\|_0\}$. The space required is $O(k\log(n)\log(\delta^{-1})(\log(\alpha) + \log\log(n)))$ bits.*

8 LOWER BOUNDS

We now show matching or nearly matching lower bounds for all problems we have considered. Our lower bounds all follow via reductions from one-way randomized communication complexity. We consider both the public coin model, where Alice and Bob are given access to infinitely many shared random bits, as well as the private coin model, where they do not have shared randomness.

We begin with the hardness of the heavy hitters problem in the strict-turnstile setting. Our hardness result holds not just for α-property streams, but even for the special case of strong α-property streams (Definition 1.2). The result matches our upper bound for normal α-property streams from Theorem 3.2 up to $\log\log(n)$ and $\log(\epsilon^{-1})$ terms.

Theorem 8.1. *For $p \geq 1$ and $\epsilon \in (0,1)$, any one-pass L_p heavy hitters algorithm for strong L_1 α-property streams in the strict turnstile model which returns a set containing all $i \in [n]$ such that $|f_i| \geq \epsilon\|f\|_p$ and no $i \in [n]$ such that $|f_i| < (\epsilon/2)\|f\|_p$ with probability at least $2/3$ requires $\Omega(\epsilon^{-p}\log(n\epsilon^p)\log(\alpha))$ bits.*

Next, we demonstrate the hardness of estimating the L_1 norm in the α-property setting. First, we show that the problem of L_1 estimation in the general turnstile model requires $\Omega(\log(n))$-bits even for α property streams with $\alpha = O(1)$. We also give a lower bound of $\Omega(1/\epsilon^2\log(\alpha))$ bits for general turnstile L_1 estimation for strong α-property streams.

Theorem 8.2. *For any $\alpha \geq 3/2$, any algorithm that produces an estimate $\tilde{L_1} \in (1 \pm 1/16)\|f\|_1$ of a general turnstile stream f with the L_1 α property with probability $2/3$ requires $\Omega(\log(n))$ bits of space.*

Theorem 8.3. *Any algorithm that produces an estimate $\tilde{L} \in (1 \pm \epsilon)\|f\|_1$ with probability $11/12$ of a general turnstile stream $f \in \mathbb{R}^n$ with the strong L_1 α property requires $\Omega(\frac{1}{\epsilon^2}\log(\epsilon^2\alpha))$ bits of space.*

We now give a matching lower bound for L_1 estimation of strict-turnstile strong α-property streams. This exactly matches our upper bound of Theorem 5.2, which is for the more general α-property setting.

Theorem 8.4. *For $\epsilon \in (0,1/2)$ and $\alpha < n$, any algorithm which gives an ϵ-relative error approximation of the L_1 of a strong L_1 α property stream in the strict turnstile setting with probability at least $2/3$ must use $\Omega(\log(\alpha) + \log(1/\epsilon) + \log\log(n))$ bits.*

We now prove a lower bound on L_0 estimation. Our lower bound matches our upper bound of Theorem 6.10 up to $\log\log(n)$ and $\log(1/\epsilon)$ multiplicative factors, and a $\log(n)$ additive term.

Theorem 8.5. *For $\epsilon \in (0,1/2)$ and $\alpha < n/\log(n)$, any one pass algorithm that gives a $(1\pm\epsilon)$ multiplicative approximation of the L_0 of a L_0 α-property stream in the strict turnstile setting with probability at least $11/12$ must use $\Omega(\epsilon^{-2}\log(\epsilon^2\alpha) + \log\log(n))$ bits of space.*

Next, we give lower bounds for L_1 and support sampling. Our lower bound for L_1 samplers holds in the more restricted strong α-property setting, and for such streams we show that even those which return an index from a distribution with variation distance at most $1/6$ from the L_1 distribution $|f_i|/\|f\|_1$ requires $\Omega(\log(n)\log(\alpha))$ bits. In this setting, taking $\epsilon = o(1)$, this bound matches our upper bound from Theorem 4.3 up to $\log\log(n)$ terms. For $\alpha = o(n)$, our support sampling lower bound matches our upper bound in Theorem 7.2.

Theorem 8.6. *Any one pass L_1 sampler of a strong L_1 α-property stream f in the strict turnstile model with an output distribution that has variation distance at most $1/6$ from the L_1 distribution $|f_i|/\|f\|_1$ and succeeds with probability $2/3$ requires $\Omega(\log(n)\log(\alpha))$ bits of space.*

Theorem 8.7. *Any one pass support sampler that outputs an arbitrary $i \in [n]$ such that $f_i \neq 0$, of an L_0 α-property stream with failure probability at most $1/3$, requires $\Omega(\log(n/\alpha)\log(\alpha))$ bits of space.*

Finally, we show that estimating inner products even for *strong* α-property streams requires $\Omega(\epsilon^{-1}\log(\alpha))$ bits of space. Setting $\alpha = n$, we obtain an $\Omega(\epsilon^{-1}\log(n))$ lower bound for unbounded deletion streams, which our upper bound beats for small α.

Theorem 8.8. *Any one pass algorithm that runs on two strong L_1 α-property streams f, g in the strict turnstile setting and computes a value $\mathrm{IP}(f,g)$ such that $\mathrm{IP}(f,g) = \langle f,g\rangle + \epsilon\|f\|_1\|g\|_1$ with probability $2/3$ requires $\Omega(\epsilon^{-1}\log(\alpha))$ bits of space.*

9 CONCLUSION

We have shown that for bounded deletion streams, many important L_0 and L_1 streaming problems can be solved more efficiently. For L_1, the fact the f_i's are approximately preserved under sampling poly(α) updates paves the way for our results, whereas for L_0 we utilizes the fact that $O(\log(\alpha))$-levels of sub-sampling are sufficient for several algorithms. Interestingly, it is unclear whether improved algorithms for α-property streams exist for L_2 problems, such as L_2 heavy hitters or L_2 estimation. The difficulty stems from the fact that $\|f\|_2$ is not preserved in any form under sampling, and thus L_2 guarantees seem to require different techniques than those used in this paper.

However, we note that by utilizing the heavy hitters algorithm of [10], one can solve the general turnstile L_2 heavy hitters problem for α-property streams in $O(\alpha^2\log(n)\log(\alpha))$ space. A proof is sketched in Appendix A. Clearly a polynomial dependence on α is not desirable; however, for the applications in which α is a constant this still represents a significant improvement over the $\Omega(\log^2(n))$ lower bound for turnstile streams. We leave it as an open question whether the optimal dependence on α can be made logarithmic.

Additionally, it is possible that problems in dynamic geometric data streams (see [33]) would benefit by bounding the number of deletions on the streams in question. We note that several of these geometric problems directly reduce to problems in the data stream model studied here, for which our results directly apply.

Finally, it would be interesting to see if analogous models with lower bounds on the "signal size" of the input would result in improved algorithms for other classes of sketching algorithms. In particular, determining the appropriate analogue of the α-property for linear algebra problems, such as row-rank approximation and regression, could be a fruitful direction for further research.

REFERENCES

[1] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.
[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2095116.2095156.
[3] Miklos Ajtai, Randal Chilton Burns, Ronald Fagin, and Larry Joseph Stockmeyer. System and method for differential compression of data from a plurality of binary sources, April 16 2002. US Patent 6,374,250.
[4] Aditya Akella, Ashwin Bharambe, Mike Reiter, and Srinivasan Seshan. Detecting ddos attacks on isp networks.
[5] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 10–20. ACM, 1999.
[6] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. *arXiv preprint arXiv:1011.1263*, 2010.

[7] Khanh Do Ba, Piotr Indyk, Eric Price, and David P Woodruff. Lower bounds for sparse recovery. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1190–1197. SIAM, 2010.

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

[9] Arnab Bhattacharyya, Palash Dey, and David P Woodruff. An optimal algorithm for l1-heavy hitters in insertion streams and related problems. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 385–400. ACM, 2016.

[10] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an l2 heavy hitters algorithm using constant memory. *arXiv preprint arXiv:1603.00759*, 2016.

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

[12] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

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

[14] Edith Cohen. Stream sampling for frequency cap statistics. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 159–168. ACM, 2015.

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

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

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

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

[19] Graham Cormode, Theodore Johnson, Flip Korn, Shan Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic udafs at streaming speeds. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 35–46. ACM, 2004.

[20] Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. *arXiv preprint arXiv:1608.03118*, 2016.

[21] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

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

[23] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 153–166. ACM, 2003.

[24] Schkolnick Finkelstein, Mario Schkolnick, and Paolo Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems (TODS)*, 13(1):91–128, 1988.

[25] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer Berlin Heidelberg, 2016. URL: https://books.google.com/books?id=qiSpDAAAQBAJ.

[26] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Data stream management: A brave new world. pages 1–9, 01 2016.

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

[28] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bernoulli samples over evolving multisets. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 93–102, 2007. URL: http://doi.acm.org/10.1145/1265530.1265544, doi:10.1145/1265530.1265544.

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

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

[31] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 241–247. ACM, 2015.

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

[33] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 373–380. ACM, 2004.

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

[35] Rajesh Jayaram and David P. Woodruff. Data streams with bounded deletions. *arXiv Preprint arXiv:1803.08777.*

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

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

[38] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

[39] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.

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

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

[42] Christian Konrad. Maximum matching in turnstile streams. In *Algorithms-ESA 2015*, pages 840–852. Springer, 2015.

[43] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

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

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

[46] David Moore, 2001. URL: http://www.caida.org/research/security/code-red/.

[47] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

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

[49] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[50] Florin Rusu and Alin Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems (TODS)*, 32(2):11, 2007.

[51] Florin Rusu and Alin Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)*, 33(3):15, 2008.

[52] Amit Shukla and Prasad Deshpande. Storage estimation for multidimensional aggregates in the presence of hierarchies.

[53] Dan Teodosiu, Nikolaj Bjorner, Yuri Gurevich, Mark Manasse, and Joe Porkka. Optimizing file replication over limited-bandwidth networks using remote differential compression. 2006.

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

[55] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast ip networks. In *Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on*, pages 172–177. IEEE, 2005.

A SKETCH OF L_2 HEAVY HITTERS ALGORITHM

We first note that the BPTree algorithm for [10] solves the L_2 heavy hitters problem in space $O(\epsilon^{-2}\log(n)\log(1/\epsilon))$ with probability 2/3. Recall that the L_2 version of the problem asks to return a set $S \subset [n]$ with all i such that $|f_i| \geq \epsilon\|f\|_2$ and no j such that $|f_j| < (\epsilon/2)\|f\|_2$. Also recall that the α property states that $\|I+D\|_2 \leq \alpha\|f\|_2$, where I is the vector of the stream restricted to the positive updates and D is the entry-wise absolute value of the vector of the stream restricted to the negative updates. It follows that if $i \in [n]$ is an ϵ heavy hitter then $\|I + D\|_2 \leq \alpha\|f\|_2 \leq \alpha/\epsilon|f_i| \leq \alpha/\epsilon|I_i + D_i|$. So in the insertion-only stream $I+D$ where every update is positive, the item i must be an ϵ/α heavy hitter.

Using this observation, we can solve the problem as follows. First run BPTree with $\epsilon' = \epsilon/\alpha$ to obtain a set $S \subset n$ with all i such that $|I_i+D_i| \geq (\epsilon/\alpha)\|I+D\|_2$ and no j such that $|I_j+D_j| < (\epsilon/2\alpha)\|I+D\|_2$. It follows that $|S| = O(\alpha^2/\epsilon^2)$. So in parallel we run an instance of Countsketch on the original stream f with $O(1/\epsilon^2)$ rows and $O(\log(\alpha/\epsilon))$ columns. For a fixed i, this gives an estimate of $|f_i|$ with additive error $(\epsilon/4)\|f\|_2$ with probability $1 - O(\epsilon/\alpha)$. At the end, we then query the Countsketch for every $i \in S$, and return only those which have estimated weight $(3\epsilon/4)\|f\|_2$. By union bounding over all $|S|$ queries, with constant probability the Countsketch will correctly identify all $i \in |S|$ that are ϵ heavy hitters, and will throw out all items in S with weight less than $(\epsilon/2)\|f\|_2$. Since all ϵ heavy hitters are in S, this solves the problem. The space to run BPTree is $O(\alpha^2/\epsilon^2\log(n)\log(\alpha/\epsilon)$, which dominates the cost of the Countsketch.