A Near-Optimal Algorithm for Computing the Entropy of a Stream

Amit Chakrabarti *
ac@cs.dartmouth.edu

Graham Cormode †
graham@research.att.com

Andrew McGregor ‡
andrewm@seas.upenn.edu

Abstract

We describe a simple algorithm for approximating the empirical entropy of a stream of m values in a single pass, using $O(\varepsilon^{-2}\log(\delta^{-1})\log m)$ words of space. Our algorithm is based upon a novel extension of a method introduced by Alon, Matias, and Szegedy [1]. We show a space lower bound of $\Omega(\varepsilon^{-2}/\log(\varepsilon^{-1}))$, meaning that our algorithm is near-optimal in terms of its dependency on ε. This improves over previous work on this problem [9, 17, 21, 6]. We show that generalizing to kth order entropy requires close to linear space for all $k \geq 1$, and give additive approximations using our algorithm. Lastly, we show how to compute a multiplicative approximation to the entropy of a random walk on an undirected graph.

1 Introduction

The problem of computing the frequency moments of a stream [1] has stimulated significant research within the algorithms community, leading to new algorithmic techniques and lower bounds. For all frequency moments, matching upper and lower bounds for the space complexity are now known [10, 25, 20, 7]. In the last year, attention has been focused on the strongly related question of computing the *entropy* of a stream. Motivated by networking applications [16, 24, 26], several partial results have been shown on computing the (empirical) entropy of a sequence of m items in sublinear space [9, 17, 21, 6]. In this paper, we show a simple algorithm for computing an (ε, δ)-approximation to this quantity in a single pass, using $O(\varepsilon^{-2}\log(\delta^{-1})\log m)$ words of space. We also show a lower bound of $\Omega(\varepsilon^{-2}/\log(\varepsilon^{-1}))$, proving that our algorithm is near-optimal in terms of its dependency on ε. We then give algorithms and lower bounds for kth order entropy, a quantity that arises in text compression, based on our results for empirical (zeroth order) entropy. We also provide algorithms to multiplicatively approximate the entropy of a random walk over an undirected graph. Our techniques are based on a method originating with Alon, Matias, and Szegedy [1]. However, this alone is insufficient to approximate the entropy in bounded space. At the core of their method is a procedure for drawing a uniform sample from the stream. We show how to extend this to drawing a larger sample, according to a specific distribution, of distinct values from the stream. The idea is straightforward to implement, and may have applications to other problems. For the estimation of entropy we will show that keeping a "back-up sample" of a single additional item is sufficient to guarantee the desired space bounds. In Section 2 we discuss this case and present our algorithm for approximating entropy (along with the lower bound.) The results pertaining to kth order entropy are in Section 3. The extension to entropy of a random walk on a graph is in Section 4.

*Work supported by an NSF CAREER award and by Dartmouth College startup funds.
†Work carried out in part while at Lucent Bell Laboratories.
‡Work partially supported by ONR N00014-04-1-0735.

Preliminaries

A randomized algorithm is said to (ε, δ)-approximate a real number Q if it outputs a value $\hat{Q}$ such that $|\hat{Q} - Q| \leq \varepsilon Q$ with probability at least $(1 - \delta)$ over its internal coin tosses. Our goal is to produce such (ε, δ)-approximations for the entropy of a stream. We first introduce some notation and definitions.

Definition 1. For a data stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, with each *token* $a_j \in [n]$, we define $m_i := |\{j : a_j = i\}|$ and $p_i := m_i/m$, for each $i \in [n]$. The *empirical probability distribution* of A is $p := (p_1, p_2, \ldots, p_n)$. The *empirical entropy* of A is defined[1] as $H(p) := \sum_{i=1}^{n} -p_i \lg p_i$. The *entropy norm* of A is $F_H := \sum_{i=1}^{n} m_i \lg m_i$.

Clearly F_H and H are closely related, since we can write $F_H = m \lg m - mH$. However, they differ significantly in their approximability: F_H cannot be approximated within constant factors in poly-logarithmic space [9], while we show here an (ε, δ)-approximation of H in poly-logarithmic space.

Prior Work

In the networking world, the problem of approximating the entropy of a stream was considered in Lall et al. [21]. They focused on estimating F_H, under assumptions about the distribution defined by the stream that ensured that computing H based on their estimate of F_H would give accurate results. Guha, McGregor and Venkatasubramanian [17] gave constant factor as well as (ε, δ)-approximations for H, using space that depends on the value of H. Chakrabarti, Do Ba and Muthukrishnan [9] gave a one pass algorithm for approximating H with sublinear but polynomial in m space, as well as a two-pass algorithm requiring only poly-logarithmic space. Most recently, Bhuvanagiri and Ganguly [6] described an algorithm that can approximate H in poly-logarithmic space in a single pass. The algorithm is based on the same ideas and techniques as recent algorithms for optimally approximating frequency moments [20, 7], and can tolerate streams in which previously observed items are removed. The exact space bound is

$$O\left(\varepsilon^{-3}(\log^4 m)(\log \delta^{-1})\frac{\log m + \log n + \log \varepsilon^{-1}}{\log \varepsilon^{-1} + \log\log m}\right) ,$$

which is suboptimal in its dependency on ε, and has high cost in terms of $\log m$.

2 Computing the Entropy of a Stream

2.1 The Main Algorithm

Consider a data stream A as in Definition 1. For a real-valued function f such that $f(0) = 0$, let us define $\overline{f}(A) := \frac{1}{m}\sum_{i=1}^{n} f(m_i)$. We base our approach on the method of Alon, Matias and Szegedy [1] to estimate quantities of the form $\overline{f}(A)$: note that the empirical entropy of A is one such quantity with $f(m_i) = m_i \log(m/m_i)$.

Definition 2. Let $\mathcal{D}(A)$ be the distribution of the random variable R defined thus: Pick $J \in [m]$ uniformly at random and let $R = |\{j : a_j = a_J,\ J \leq j \leq m\}|$.

[1] Here and throughout we use $\lg x$ to denote $\log_2 x$.

The core idea is to space-efficiently generate a random variable $R \sim \mathcal{D}(A)$. For an integer c, define the random variable

$$\mathrm{Est}_f(R, c) \ := \ \frac{1}{c}\sum_{i=1}^{c} X_i \ , \tag{1}$$

where the random variables $\{X_i\}$ are independent and each distributed identically to $(f(R)-f(R-1))$. Appealing to Chernoff-Hoeffding bounds one can show that by increasing c, $\mathrm{Est}_f(R, c)$ can be made arbitrarily close to $\overline{f}(A)$. This is formalized in the lemma below.

Lemma 1. *Let* $X := f(R) - f(R-1)$*,* $a, b \geq 0$ *such that* $-a \leq X \leq b$*, and*

$$c \geq 3(1 + a/\,\mathrm{E}[X])^2 \varepsilon^{-2} \ln(2\delta^{-1})(a+b)/(a+\mathrm{E}[X]) \ .$$

Then $\mathrm{E}[X] = \overline{f}(A)$ *and, if* $\mathrm{E}[X] \geq 0$*, the estimator* $\mathrm{Est}_f(R, c)$ *gives an* (ε, δ)*-approximation to* $\overline{f}(A)$ *using space c times the space required to maintain R.*

Proof. $\mathrm{E}[X] = \overline{f}(A)$ follows by straightforward calculation of the expectation. Consider the random variable $Y := (X+a)/(a+b)$. First note that $Y \in [0, 1]$. Therefore Chernoff-Hoeffding bounds imply that, if $\{Y_i\}$ are independent and each distributed identically to Y, $\mathrm{Est}'_f(R, c) = c^{-1}\sum_{i\in[c]} Y_i$ is an $(\varepsilon/(1 + a/\,\mathrm{E}[X]), \delta)$-approximation to $(\overline{f}(A) + a)/(a+b)$. Note that, $\mathrm{Est}'_f(R, c) = \left(\mathrm{Est}_f(R, c) + a\right)/(a+b)$. This implies that,

$$\begin{aligned}
\Pr\left[|\,\mathrm{Est}_f(R, c) - \overline{f}(A)| > \varepsilon \overline{f}(A)\right] \ &= \ \Pr\left[\left|(a+b)\,\mathrm{Est}'_f(R, c) - \overline{f}(A) - a\right| > \varepsilon \overline{f}(A)\right] \\
&= \ \Pr\left[\left|\mathrm{Est}'_f(R, c) - \frac{\overline{f}(A)+a}{a+b}\right| > \frac{\varepsilon}{1 + a/\,\mathrm{E}[X]}\frac{\overline{f}(A)+a}{a+b}\right] \\
&\leq \ \delta \ .
\end{aligned}$$

Therefore, $\mathrm{Est}_f(R, c)$ gives an (ε, δ)-approximation to $\overline{f}(A)$ as claimed. □

Overview of the technique

We now give some of the intuition behind our algorithm for estimating $H(p)$. Let A' denote the substream of A obtained by removing from A all occurrences of the most frequent token (with ties broken arbitrarily) and let $R' \sim \mathcal{D}(A')$. A key component of our algorithm (see Algorithm *Maintain-Samples* below) is a technique to simultaneously maintain R and enough extra information that lets us recover R' when we need it. Let $p_{\max} := \max_i p_i$. Let λ be given by

$$\lambda(x) \ := \ x \lg(m/x)\,, \ \text{where } \lambda(0) \ := \ 0\,, \tag{2}$$

so that $\overline{\lambda}(A) = H(p)$. Define $X = \lambda(R) - \lambda(R-1)$ and $X' = \lambda(R') - \lambda(R'-1)$. If $p_{\max}$ is bounded away from 1 then we can show that $1/\,\mathrm{E}[X]$ is "small," so $\mathrm{Est}_\lambda(R, c)$ gives us our desired estimator for a "small" value of c, by Lemma 1. If, on the other hand, $p_{\max} > \frac{1}{2}$ then we can recover R' and can show that $1/\,\mathrm{E}[X']$ is "small." Finally, by our analysis we can show that $\mathrm{Est}_\lambda(R', c)$ and an estimate of $p_{\max}$ can be combined to give an (ε, δ)-approximation to $H(p)$. This logic is given in Algorithm *Entropy-Estimator* below.

Thus, our algorithm must also maintain an estimate of $p_{\max}$ in parallel to Algorithm *Maintain-Samples*. There are a number of ways of doing this and here we choose to use the Misra-Gries algorithm [22] with a sufficiently large number of counters. This (deterministic) algorithm takes a parameter k — the number of counters — and processes the stream, retaining up to k pairs $(i, \hat{m}_i)$, where i is a token and the counter $\hat{m}_i$ is an estimate of its frequency m_i. The algorithm starts out holding no pairs and implicitly setting each $\hat{m}_i = 0$. Upon reading a token, i, if a pair $(i, \hat{m}_i)$ has already been retained, then $\hat{m}_i$ is incremented; else, if fewer than k pairs have been retained, then a new pair $(i, 1)$ is created and retained; else, $\hat{m}_j$ is decremented for each retained pair $(j, \hat{m}_j)$ and then all pairs of the form $(j, 0)$ are discarded. The following lemma summarizes the key properties of this algorithm; the proof is simple (see, e.g., [8]) and we skip it below.

Lemma 2. *The estimates $\hat{m}_i$ computed by the Misra-Gries algorithm using k counters satisfy $\hat{m}_i \le m_i$ and $m_i - \hat{m}_i \le (m - m_i)/k$.* □

We now describe our algorithm more precisely with some pseudocode. By abuse of notation we use $\mathrm{Est}_\lambda(r, c)$ to also denote the algorithmic procedure of running in parallel c copies of an algorithm that produces r and combining these results as in (1).

Algorithm *Maintain-Samples*

for $a \in A$
    do Let t be a random number in the range $[m^3]$
        if $a = s_0$
          then if $t < t_0$ **then** $(s_0, t_0, r_0) \leftarrow (a, t, 1)$ **else** $r_0 \leftarrow r_0 + 1$
          else if $a = s_1$ **then** $r_1 \leftarrow r_1 + 1$
              if $t < t_0$
                then $(s_1, t_1, r_1) \leftarrow (s_0, t_0, r_0)$; $(s_0, t_0, r_0) \leftarrow (a, t, 1)$
                else if $t < t_1$ **then** $(s_1, t_1, r_1) \leftarrow (a, t, 1)$

Algorithm *Entropy-Estimator*

$c \leftarrow 16\varepsilon^{-2} \ln(2\delta^{-1}) \lg(me)$
Run the Misra-Gries algorithm on A with $k = \lceil 7\varepsilon^{-1} \rceil$ counters, in parallel with *Maintain-Samples*
if Misra-Gries retains a token i with counter $\hat{m}_i > m/2$
    then $(i_{\max}, \hat{p}_{\max}) \leftarrow (i, \hat{m}_i/m)$
        if $a_0 = i_{\max}$ **then** $r \leftarrow r_1$ **else** $r \leftarrow r_0$
        return $(1 - \hat{p}_{\max}) \cdot \mathrm{Est}_\lambda(r, c) + \hat{p}_{\max} \lg(1/\hat{p}_{\max})$
    else **return** $\mathrm{Est}_\lambda(r_0, c)$

Figure 1: Algorithms for sampling and estimating entropy.

Maintaining Samples from the Stream

We show a procedure that allows us to generate R and R' with the appropriate distributions. For each token a in the stream, we draw t, a random number in the range $[m^3]$, as its label. We choose to store certain tokens from the stream, along with their label and the count of the number of times the same token has been observed in the stream since it was last picked. We store *two* such tokens: the token s_0 that has achieved the least t value seen so far, and the token s_1 such that it has the least t value of all tokens not equal to s_0 seen so far. Let t_0 and t_1 denote their corresponding labels, and let r_0 and r_1 denote their counts in the above sense. Note that it is easy to maintain these properties as new items arrive in the stream, as Algorithm *Maintain-Samples* illustrates.

Lemma 3. *Algorithm Maintain-Samples satisfies the following properties. (i) After processing the whole stream A, s_0 is picked uniformly at random from A and $r_0 \sim \mathcal{D}(A)$. (ii) For $a \in [n]$, let $A \setminus a$ denote the stream A with all occurrences of a removed. Suppose we set s and r thus: if $s_0 \neq a$ then $s = s_0$ and $r = r_0$, else $s = s_1$ and $r = r_1$. Then s is picked uniformly from $A \setminus a$ and $r \sim \mathcal{D}(A \setminus a)$.*

Proof. To prove (i), note that the way we pick each label t ensures that (w.h.p.) there are no collisions amongst labels and, conditioned on this, the probability that any particular token gets the lowest label value is $1/m$.

We show (ii) by reducing to the previous case. Imagine generating the stream $A \setminus a$ and running the algorithm on it. Clearly, picking the item with the smallest t value samples uniformly from $A \setminus a$. Now let us add back in all the occurrences of a from A. One of these may achieve a lower t value than any item in $A \setminus a$, in which case it will be picked as s_0, but then s_1 will correspond to the sample we wanted from $A \setminus a$, so we can return that. Else, $s_0 \neq a$, and is a uniform sample from $A \setminus a$. Hence, by checking whether $s_0 = a$ or not, we can choose a uniform sample from $A \setminus a$. The claim about the distribution of r is now straightforward: we only need to observe from the pseudocode that, for $j \in \{0, 1\}$, r_j correctly counts the number of occurrences of s_j in A from the time s_j was last picked. □

Analysis of the Algorithm

We now analyse our main algorithm, given in full in Algorithm *Entropy-Estimator*.

Theorem 4. *Algorithm Entropy-Estimator uses space $O(\varepsilon^{-2} \log(\delta^{-1}) \log m (\log m + \log n))$ bits and gives an (ε, δ)-approximation to $H(p)$.*

Proof. To argue about the correctness of Algorithm *Entropy-Estimator*, we first look closely at the Misra-Gries algorithm used within it. By Lemma 2, $\hat{p}_i := \hat{m}_i/m$ is a good estimate of p_i. To be precise, $|\hat{p}_i - p_i| \leq (1 - p_i)/k$. Hence, by virtue of the estimation method, if $p_i > \frac{2}{3}$ and $k \geq 2$, then i must be among the tokens retained and must satisfy $\hat{p}_i > \frac{1}{2}$. Therefore, in this case we will pick $i_{\max}$ — the item with maximum frequency — correctly, and $p_{\max}$ will satisfy

$$\hat{p}_{\max} \leq p_{\max} \quad \text{and} \quad |\hat{p}_{\max} - p_{\max}| \leq \frac{1 - p_{\max}}{k} . \tag{3}$$

Let A, A', R, R', X, X' be as before. Suppose $\hat{p}_{\max} \leq \frac{1}{2}$. The algorithm then reaches Line 7. By Part (i) of Lemma 3, the returned value is $\mathrm{Est}_\lambda(R, c)$. Now (3), together with $k \geq 2$, implies $p_{\max} \leq \frac{2}{3}$ and a simple convexity argument shows that $H(p) \geq \frac{2}{3} \lg \frac{3}{2} + \frac{1}{3} \lg \frac{3}{1} > 0.9$. Note that $-\lg e \leq X \leq \lg m$. This follows because,

$$\frac{d}{dx} x \lg\left(\frac{m}{x}\right) = \lg\left(\frac{m}{x}\right) - \lg e$$

and, using the Mean Value Theorem,

$$\forall r \in [m], \quad \inf_{x \in [r-1, r]} \lambda'(x) \leq \lambda(r) - \lambda(r-1) \leq \sup_{x \in [r-1, r]} \lambda'(x) \quad .$$

Consequently,

$$X \leq \max\{\lambda(1) - \lambda(0), \max_{x \in \{2, \ldots, m\}} (\lambda(r) - \lambda(r-1))\} \leq \max\{\lg m, \sup_{x \in [1, m]} (\lg(m/x) - \lg e)\} \leq \lg m$$

and

$$X \geq \inf_{x \in [0,m]} (\lg(m/x) - \lg e) = -\lg e \ .$$

Hence Lemma 1 implies that c is large enough to ensure that the return value is a $(\frac{3}{4}\varepsilon, \delta)$-approximation to $H(p)$.

Now suppose $\hat{p}_{\max} > \frac{1}{2}$. The algorithm then reaches Line 6. By Part (ii) of Lemma 3, the return value is $(1 - \hat{p}_{\max}) \cdot \mathrm{Est}_\lambda(R', c) + \hat{p}_{\max} \lg(1/\hat{p}_{\max})$, and (3) implies that $p_{\max} > \frac{1}{2}$. Assume, w.l.o.g., that $i_{\max} = 1$. Then

$$\mathrm{E}[X'] \ = \ \overline{\lambda}(A') \ = \ \frac{1}{m - m_1} \sum_{i=2}^{n} \lambda(m_i) \ \geq \ \lg \frac{m}{m - m_1} \ \geq \ 1 \,,$$

where the penultimate inequality follows by convexity arguments. As before, $-\lg e \leq X \leq \lg m$, and hence Lemma 1 implies that c is large enough to ensure that $\mathrm{Est}_\lambda(R', c)$ is a $(\frac{3}{4}\varepsilon, \delta)$-approximation to $\overline{\lambda}(A')$.

Next, we show that $\hat{p}_1 \lg(1/\hat{p}_1)$ is a $(\frac{2}{k}, 0)$-approximation to $p_1 \lg(1/p_1)$, as follows:

$$\frac{|p_1 \lg(1/p_1) - \hat{p}_1 \lg(1/\hat{p}_1)|}{p_1 \lg(1/p_1)} \ \leq \ \frac{|\hat{p}_1 - p_1|}{p_1 \lg(1/p_1)} \max_{p \in [\frac{1}{2}, 1]} \left| \frac{d}{dp} (p \lg(1/p)) \right| \ \leq \ \frac{(1 - p_1)}{k \, p_1 \lg(1/p_1)} \cdot \lg e \ \leq \ \frac{2}{k} \,,$$

where the final inequality follows from the fact that $g(p) := (1 - p)/(p \ln(1/p))$ is non-increasing in the interval $[\frac{1}{2}, 1]$, so $g(p) \leq g(\frac{1}{2}) < 2$. To see this, note that $1 - p + \ln p \leq 0$ for all positive p and that $g'(p) = (1 - p + \ln p)/(p \ln p)^2$. Now observe that

$$H(p) \ = \ (1 - p_1)\overline{\lambda}(A') + p_1 \lg(1/p_1) \,. \tag{4}$$

From (3) it follows that $(1 - \hat{p}_1)$ is an $(\frac{1}{k}, 0)$-approximation to $(1 - p_1)$. Setting $k \geq \lceil 7\varepsilon^{-1} \rceil$, and assuming $\varepsilon \leq 1$ ensures that that $(1 - \hat{p}_1) \cdot \mathrm{Est}_\lambda(R', c)$ is a (ε, δ)-approximation to $(1 - p_1)\overline{\lambda}(A')$, and $\hat{p}_1 \lg(1/\hat{p}_1)$ is a (better than) $(\varepsilon, 0)$-approximation to $p_1 \lg(1/p_1)$. Thus, we have shown that in this case the algorithm returns a (ε, δ)-approximation to $H(p)$, since both terms in (4) are approximated with relative error.

The claim about the space usage is straightforward. The Misra-Gries algorithm requires $O(k) = O(\varepsilon^{-1})$ counters and item identifiers. Each run of Algorithm *Maintain-Samples* requires $O(1)$ counters, labels, and item identifiers, and there are $c = O(\varepsilon^{-2} \log(\delta^{-1}) \log m)$ such runs. Everything stored is either an item from the stream, a counter that is bounded by m, or a label that is bounded by m^3, so the space for each of these is $O(\log m + \log n)$ bits. $\square$

2.2 Variations on the Algorithm

Randomness and Stream Length

As described, our algorithm requires $O(m \log m)$ bits of randomness, since we require a random number in the range $[m^3]$ for each item in the stream. This randomness requirement can be reduced to $O(\log^{O(1)} m)$ bits by standard arguments invoking Nisan's pseudorandom generator [23]. An alternate approach is to use a hash function from a min-wise independent family on the stream index to generate t [18]. This requires a modification to the analysis: the probability of picking any fixed item changes from $1/m$ to a value in the interval $[(1 - \varepsilon)/m, (1 + \varepsilon)/m]$. One can show that this introduces a $1 + O(\varepsilon)$ factor in the expressions for expectation and variance of the estimators, which does not affect the overall correctness; an additional $O(\log n \log \varepsilon^{-1})$ factor in space would also be incurred to store the descriptions of the hash functions.

The algorithm above also seems to require prior knowledge of m, although an upper bound clearly suffices (we can compute the true m as the stream arrives). But we only need to know m in order to choose the size of the random labels large enough to avoid collisions. Should the assumed bound be proven too low, it suffices to extend the length of labels t_0 and t_1 by drawing further random bits in the event of collisions to break ties. Invoking the principle of deferred decisions, it is clear that the correctness of the algorithm is unaffected.

Sliding Window Computations

In many cases it is desirable to compute functions not over the whole semi-infinite stream, but rather over a sliding window of the last W updates. Our method accommodates such an extension with an $O(\log^2 W)$ expansion of space (with high probability). Formally, define the sliding window count of i as $m_i^w = |\{j | a_j = i, i > m - w\}|$. The empirical probability is $p_i^w = m_i^w / w$, and the empirical entropy is $H(p^w) = \sum_{i=1}^n -p_i^w \lg p_i^w$.

Lemma 5. *We can approximate $H(p^w)$ for any $w < W$ in space bounded by $O(\varepsilon^{-2} \log(\delta^{-1}) \log^3 W)$ machine words with high probability.*

Proof. We present an algorithm that retains sufficient information so that, after observing the stream of values, given $w < W$ we can recover the information that Algorithm *Entropy-Estimator* would have stored had only the most recent w values been presented to it. From this, the correctness follows immediately. Thus, we must be able to compute $s_0^w, r_0^w, s_1^w, r_1^w, i_{\max}^w$ and $p_{\max}^w$, the values of $s_0, r_0, s_1, r_1, i_{\max}$ and $p_{\max}$ on the substreams defined by the sliding window specified by w.

For $i_{\max}^w$ and $p_{\max}^w$, it is not sufficient to apply standard sliding window frequent items queries [2]. To provide the overall accuracy guarantee, we needed to approximate $p_{\max}$ with error proportion to $\varepsilon'(1 - p_{\max}^w)$ for a parameter ε'. Prior work gives guarantees only in terms of $\varepsilon' p_1^w$, so we need to adopt a new approach. We replace our use of the Misra-Gries algorithm with the Count-Min sketch [11]. This is a randomized algorithm that hashes each input item to $O(\log \delta^{-1})$ buckets, and maintains a sum of counts within each of a total of $O(\varepsilon^{-1} \log \delta^{-1})$ buckets. If we were able to create a CM-sketch summarizing just the most recent w updates, then we would be able to find an (ε, δ) approximation to $(1 - p_{\max}^w)$, and hence also find $p_{\max}^w$ with error $\varepsilon(1 - p_{\max}^w)$. This follow immediately from the properties of the sketch proved in [11]. In order to make this valid for arbitrary sliding windows, we replace each counter within the sketch with an Exponential Histogram or Determinsitic Wave data structure [13, 15]. This allows us to $(\varepsilon, 0)$ approximate the count of each bucket within the most recent $w < W$ timesteps in space $O(\varepsilon^{-1} \log^2 W)$. Combining these and rescaling ε, one can build an (ε, δ) approximation of $(1 - p_{\max}^w)$ for any $w < W$. The space required for this estimation is $O(\varepsilon^{-2} \log^2 W \log \delta^{-1} (\log m + \log n))$ bits.

For s_0^w, r_0^w, s_1^w and r_1^w, we can take advantage of the fact that these are defined by randomly chosen tags t_0^w and t_1^w, and for any W there are relatively few possible candidates for all the $w < W$. Let $t[j]$ be the random tag for the jth item in the stream. We maintain the following set of tuples,

$$S_0 = \{(j, a_j, t[j], r[j]) : j = \operatorname*{argmin}_{m-w<i\le m} t[j], r[j] = |\{k | a_k = a_j, k \ge j\}|, w < W\}$$

This set defines $j_0^w = \operatorname{argmin}_{m-w<i\le m} t[j]$ for $w < W$. We maintain a second set of tuples,

$$S_1 = \{(j, a_j, t[j], r[j]) : j = \operatorname*{argmin}_{i \ne j_0^w, m-w<i\le m} t[j], r[j] = |\{k | a_k = a_j, k \ge j\}|, w < W\}$$

and this set defines $j_1^w = \operatorname{argmin}_{m-w<i\le m} t[j]$ for $w < W$. Note that it is straight-forward to maintain S_0 and S_1. Then, for any $w < W$, we set,

$$(s_0^w, r_0^w) \leftarrow (a_{j_0^w}, r[j_0^w]) \quad \text{and} \quad (s_1^w, r_1^w) \leftarrow (a_{j_1^w}, r[j_1^w]) \ .$$

We now bound the sizes of S_0 and S_1. The size of S_0 can be bounded by observing that if we build a treap over the sequence of timestamp, label pairs where we order by timestamp and heapify by label, the members of S_0 correspond to precisely the right spine of the treap. As argued in [3], this approach yields a strong bound on $|S_0|$, since with high probability the height of a treap with randomly chosen priorities such as these (i.e. a random binary search tree) is logarithmic. Further, we can observe that members of S_1 correspond to nodes in the treap that are left children of members of S_0, and their right descendants. Thus, if the treap has height h, the size of S_1 is $O(h^2)$. For windows of size at most W, the implicit treap has height $O(\log W)$ with high probability.

Thus, we need to store a factor of $O(\log^2 W)$ more information for each instance of the basic estimator. The total space bound is therefore $O(\varepsilon^{-2}\log(\delta^{-1})\log^3 W(\log m + \log n))$ bits. □

2.3 Extensions to the Technique

We observe that the method we have introduced here, of allowing a sample to be drawn from a modified stream with an item removed may have other applications. The method naturally extends to allowing us to specify a set of k items to remove from the stream after the fact, by keeping the $k+1$ distinct items achieving the smallest label values. In particular, Lemma 3 can be extended to give the following.

Lemma 6. *There exists an algorithm $\mathcal{A}$, using $O(k)$ space, that returns k pairs $(s_i, r_i)_{i\in[k+1]}$ such that s_i is picked uniformly at random from $A \setminus \{s_1, \ldots, s_{i-1}\}$ and $r \sim \mathcal{D}(A \setminus \{s_1, \ldots, s_{i-1}\})$. Consequently, given a set S of size at most k and the output of $\mathcal{A}$ it is possible to sample (s, r) such that s is picked uniformly at random from $A \setminus S$ and $r \sim \mathcal{D}(A \setminus S)$.*

This may be of use in applications where we can independently identify "junk" items or other undesirable values which would dominate the stream if not removed. For example, in the case in which we wish to compute the quantiles of a distribution after removing the k most frequent items from the distribution. Additionally, the procedure may have utility in situations where a small fraction of values in the stream can significantly contribute to the variance of other estimators.

2.4 Lower Bound

We now show that the dependence of the above space bound on ε is nearly tight. To be precise, we prove the following theorem.

Theorem 7. *Any one-pass randomized $(\varepsilon, \frac{1}{4})$-approximation for $H(p)$ requires $\Omega(\varepsilon^{-2}/\log(\varepsilon^{-1}))$ space.*

Proof. Let GAP-HAMDIST denote the following (one-way) communication problem. Alice receives $x \in \{0,1\}^N$ and Bob receives $y \in \{0,1\}^N$. Alice must send a message to Bob after which Bob must answer "near" if the Hamming distance $\|x-y\|_1 \le N/2$ and "far" if $\|x-y\|_1 \ge N/2+\sqrt{N}$. They may answer arbitrarily if neither of these two cases hold. The two players may follow a randomized protocol that must work correctly with probability at least $\frac{3}{4}$. It is known [19] that GAP-HAMDIST has one-way communication complexity $\Omega(N)$.

We now reduce GAP-HAMDIST to the problem of approximating $H(p)$. Suppose $\mathcal{A}$ is a one-pass algorithm that (ε, δ)-approximates $H(p)$. Let N be chosen such that $\varepsilon^{-1} = 3\sqrt{N}\lg N$ and assume, w.l.o.g., that N is an integer. Alice and Bob will run $\mathcal{A}$ on a stream of tokens from $[N] \times \{0,1\}$ as follows. Alice feeds the stream $\langle (i, x_i)\rangle_{i=1}^N$ into $\mathcal{A}$ and then sends over the memory contents of $\mathcal{A}$ to Bob who then continues the run by feeding in the stream $\langle (i, y_i) \rangle_{i=1}^{N}$. Bob then looks at the output out($\mathcal{A}$) and answers "near" if

$$\text{out}(\mathcal{A}) \; < \; \lg N + \frac{1}{2} + \frac{1}{2\sqrt{N}}$$

and answers "far" otherwise. We now prove the correctness of this protocol.

Let $d := \|x - y\|_1$. Note that the stream constructed by Alice and Bob in the protocol will have $N - d$ tokens with frequency 2 each and $2d$ tokens with frequency 1 each. Therefore,

$$H(p) \; = \; (N-d) \cdot \frac{2}{2N} \lg \frac{2N}{2} + 2d \cdot \frac{1}{2N} \lg \frac{2N}{1} \; = \; \lg N + \frac{d}{N} \, .$$

Therefore, if $d \le N/2$, then $H(p) \le \lg N + \frac{1}{2}$ whence, with probability at least $\frac{3}{4}$, we will have

$$\text{out}(\mathcal{A}) \; \le \; (1+\varepsilon) H(p) \; \le \; \left(1 + \frac{1}{3\sqrt{N} \lg N}\right) \left(\lg N + \frac{1}{2}\right) \; < \; \lg N + \frac{1}{2} + \frac{1}{2\sqrt{N}}$$

and Bob will correctly answer "near." A similar calculation shows that if $d \ge N/2 + \sqrt{N}$ then, with probability at least $\frac{3}{4}$, Bob will correctly answer "far." Therefore the protocol is correct and the communication complexity lower bound implies that $\mathcal{A}$ must use space at least $\Omega(N) = \Omega(\varepsilon^{-2}/\log(\varepsilon^{-1}))$.

□

3 Higher-Order Entropy

The kth order entropy is a quantity defined on a sequence that quantifies how easy it is to predict a character of the sequence given the previous k characters. We start with a formal definition.

Definition 3. For a data stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, with each token $a_j \in [n]$, we define

$$m_{i_1 i_2 \ldots i_t} \; := \; |\{ j \le m - k : \, a_{j-1+l} = i_l \text{ for } l \in [t] \}| \, ; \qquad p_{i_t | i_1, i_2, \ldots, i_{t-1}} \; := \; m_{i_1 i_2 \ldots i_t} / m_{i_1 i_2 \ldots i_{t-1}} \, ,$$

for $i_1, i_2, \ldots, i_t \in [n]$. The (empirical) *kth order entropy* of A is defined as

$$H_k(A) := -\sum_{i_1} p_{i_1} \sum_{i_2} p_{i_2|i_1} \ldots \sum_{i_{k+1}} p_{i_{k+1}|i_1 \ldots i_k} \lg p_{i_{k+1}|i_1 \ldots i_k} \, .$$

Unfortunately, unlike empirical entropy, H_0, there is no small space algorithm for multiplicatively approximating H_k. This is even the case for H_1 as substantiated in the following theorem.

Theorem 8. *Approximating $H_1(A)$ up to any multiplicative error requires $\Omega(m/\log m)$ space.*

Proof. Let PREFIX denote the following (one-way) communication problem. Alice has a string $x \in \{0,1\}^N$ and Bob has a string $y \in \{0,1\}^{N'}$ with $N' \le N$. Alice must send a message to Bob, and Bob must answer "yes" if y is a prefix of x, and "no" otherwise. The one-way probabilistic communication complexity of PREFIX is $\Omega(N/\log N)$, as the following argument shows. Suppose we could solve PREFIX using C bits of communication. Repeating such a protocol $O(\log n)$ times in parallel reduces the probability of failure from constant to $O(1/n)$. But by posing $O(n)$ PREFIX queries in response to Alice's message in this latter protocol, Bob could learn x with failure probability at most a constant. Therefore, we must have $C \log n = \Omega(n)$.

Consider an instance (x, y) of PREFIX. Let Alice and Bob jointly construct the stream $A = \langle a_1, a_2, \ldots, a_N, b_1, b_2, \ldots, b_{N'}\rangle$, where $a_i = (i, x_i)$ for $i \in [N]$ and $b_i = (i, y_i)$ for $i \in [N']$. Note that,

$$H_1(A) \;=\; -\sum_i p_i \sum_j p_{j|i} \lg p_{j|i} \;=\; 0$$

if x is a prefix of y. But $H_1(A) \neq 0$ if x is not a prefix of y. This reduction proves that any multiplicative approximation to H_1 requires $\Omega(N/\log N)$ space, using the same logic as that in the conclusion of the proof of Theorem 7. Since the stream length $m = N+N' = \Theta(N)$, this translates to an $\Omega(m/\log m)$ lower bound. □

Since the above theorem effectively rules out efficient multiplicative approximation, we now turn our attention to additive approximation. The next theorem (and its proof) shows how the algorithm in Section 2 gives rise to an efficient algorithm that additively approximates the kth order entropy.

Theorem 9. *$H_k(A)$ can be ε-additively approximated with $O(k^2\varepsilon^{-2}\log(\delta^{-1})\log^2 n \log^2 m)$ space.*

Proof. We first rewrite the kth order entropy as follows.

$$\begin{aligned}
H_k(A) &= -\sum_{i_1,i_2,\ldots,i_{k+1}} p_{i_1} p_{i_2|i_1} \ldots p_{i_{k+1}|i_1 i_2\ldots i_k} \lg p_{i_{k+1}|i_1 i_2\ldots i_k} \\
&= \sum_{i_1,i_2,\ldots,i_{k+1}} \frac{m_{i_1\ldots i_{k+1}}}{m-k} \lg \frac{m_{i_1\ldots i_k}}{m_{i_1\ldots i_{k+1}}} \\
&= -\sum_{i_1,i_2,\ldots,i_k} \frac{m_{i_1\ldots i_k}}{m-k} \lg \frac{m-k}{m_{i_1\ldots i_k}} + \sum_{i_1,i_2,\ldots,i_{k+1}} \frac{m_{i_1\ldots i_{k+1}}}{m-k} \lg \frac{m-k}{m_{i_1\ldots i_{k+1}}} \\
&= H(p^{k+1}) - H(p^k)
\end{aligned}$$

where p^k is the distribution over n^k points with $p^k_{i_1 i_2\ldots i_k} = m_{i_1 i_2\ldots i_k}/(m-k)$ and p^{k+1} is the distribution over n^{k+1} points with $p^k_{i_1 i_2\ldots i_{k+1}} = m_{i_1 i_2\ldots i_{k+1}}/(m-k)$. Since $H(p^k)$ is less than $k \lg n$, if we approximate it to a multiplicative factor of at most $(1+\varepsilon/(2k\lg n))$ then we have an additive $\varepsilon/2$ approximation. Appealing to Theorem 4 this can be done in $O(k^2\varepsilon^{-2}\log(\delta^{-1})\log^2(n)\log(m))$ space. We can deal with $H(p^{k+1})$ similarly and hence we get an ε additive approximation for $H_k(A)$. Directly implementing these algorithms, we need to store strings of k characters from the input stream as a single kth order character; for large k, we can hash these strings onto the range $[m^2]$. Since there are only $m-k$ substrings of length k, then there are no collisions in this hashing w.h.p., and the space needed is only $O(\log m)$ bits for each stored item or counter. □

4 Entropy of a Random Walk

In Theorem 8 we showed that it was impossible to multiplicatively approximate the first order entropy, H_1, of a stream in sub-linear space. In this section we consider a related quantity H_G, the *unbiased random walk entropy*. We will discuss the nature of this relationship after a formal definition.

Definition 4. For a data stream $A = \langle a_1, a_2, \ldots, a_m\rangle$, with each token $a_j \in [n]$, we define an undirected graph $G(V, E)$ on n vertices, where $V = [n]$ and $E = \{\{u, v\} \in [n]^2 : u = a_j, v = a_{j+1} \text{ for some } j \in [m-1]\}$. Let d_i be the degree of node i. Then the *unbiased random walk entropy* of A is defined as,

$$H_G \;:=\; \frac{1}{2|E|}\sum_{i\in[n]} d_i \lg d_i \;.$$

Consider a stream formed by an unbiased random walk on an undirected graph G, i.e., if $a_i = j$ then a_{i+1} is uniformally chosen from the d_j neighbors of j. Then H_G is the limit of $H_1(A)$ as the length of this random walk tends to infinity:

$$H_G = \frac{1}{2|E|}\sum_{i\in[n]} d_i \lg d_i = \lim_{m\to\infty}\sum_{i\in[n]}\frac{m_i}{m}\sum_{j\in[n]}\frac{m_{ij}}{m_i}\lg\frac{m_i}{m_{ij}} = \lim_{m\to\infty} H_1(\langle a_1, a_2, \ldots, a_m\rangle),$$

since $\lim_{m\to\infty}(m_{ij}/m_i) = 1/d_i$ and $\lim_{m\to\infty}(m_i/m) = d_i/(2|E|)$ as the stationary distribution of a random walk on an undirected graph is $(d_1/(2|E|), d_2/(2|E|), \ldots, d_n/(2|E|))$.

For the rest of this section it will be convenient to reason about a stream E' that can be easily transduced from A. E' will consist of $m-1$, not necessarily distinct, edges on the set of nodes $V = [n]$, $E' = \langle e_1, e_2, \ldots, e_{m-1}\rangle$ where $e_i = (a_i, a_{i+1})$. Note that E is the set produced by removing all duplicate edges in E'.

Overview of the algorithm

Our algorithm uses the standard AMS-Estimator as described in Section 2. However, because E' includes duplicate items which we wish to disregard, our basic estimator is necessarily more complicated. The algorithm combines ideas from multi-graph streaming [12] and entropy-norm estimation [9] and uses min-wise hashing [18] and distinct element estimators [4].

Ideally the basic estimator would sample a node w uniformly from the multi-set in which each node u occurs d_u times. Then let r be uniformly chosen from $\{1, \ldots, d_w\}$. If the basic estimator were to return $g(r) = f(r) - f(r-1)$ where $f(x) = x \lg x$ then the estimator would be correct in expectation:

$$\sum_{w\in[n]}\frac{d_w}{2|E|}\sum_{r\in[d_w]}\frac{1}{d_w}(f(r) - f(r-1)) = \frac{1}{2|E|}\sum_{w\in[n]} d_w \lg d_w \ .$$

To mimic this sampling procedure we use an ε-min-wise hash function h [18] to map the distinct edges in E' into $[m]$. It allows us to pick an edge $e = (u, v)$ (almost) uniformly at random from E by finding the edge e that minimizes $h(e)$. We pick w uniformly from $\{u, v\}$. Note that w has been chosen with probability proportional to $(1 \pm \varepsilon)\frac{d_w}{2|E|}$. Let $i = \max\{j : e_j = e\}$ and consider the r distinct edges among $\{e_i, \ldots, e_m\}$ that are incident on w. Let $e^1, \ldots, e^{d_w}$ be the d_w edges that are incident on w and let $i_k = \max\{j : e_j = e^k\}$ for $k \in [d_w]$. Then r is distributed as $|\{k : i_k \geq i\}|$ and hence takes a value from $\{1, \ldots, d_w\}$ with probability $(1 \pm \varepsilon)/d_w$.

Unfortunately we cannot compute r exactly unless it is small. If $r \leq \varepsilon^{-2}$ then we maintain an exact count, by keeping the set of distinct edges. Otherwise we compute an (ε, δ)-approximation of r using a distinct element estimation algorithm, e.g. [4]. Note that if this is greater than n we replace the estimate by n to get a better bound. This will be important when bounding the maximum value of the estimator. Either way, let this (approximate) count be $\tilde{r}$. We then return $g(\tilde{r})$. The next lemma demonstrates that using $g(\tilde{r})$ rather than $g(r)$ only incurs a small amount of additional error.

Lemma 10. *Assuming $\varepsilon < 1/4$, $|g(r) - g(\tilde{r})| \leq O(\varepsilon)g(r)$ with probability at least $1 - \delta$.*

Proof. If $r \leq \varepsilon^{-2}$, then $r = \tilde{r}$, and the claim follows immediately. Therefore we focus on the case where $r > \varepsilon^{-2}$. Let $\tilde{r} = (1+\gamma)r$ where $|\gamma| \leq \varepsilon$. We write $g(r)$ as the sum of the two positive terms,

$$g(r) = \lg(r-1) + r\lg(1 + 1/(r-1))$$

and will consider the two terms in the above expression separately.

Note that for $r \geq 2$, $\frac{\tilde{r}-1}{r-1} = 1 \pm 2\varepsilon$. Hence, for the first term, and providing the distinct element estimation succeeds with its accuracy bounds,

$$|\lg(\tilde{r}-1) - \lg(r-1)| \;=\; \left|\lg \frac{\tilde{r}-1}{r-1}\right| \;=\; O(\varepsilon) \;\leq\; O(\varepsilon)\lg(r-1)\,.$$

where the last inequality follows since $r > \varepsilon^{-2}$, $\varepsilon < \frac{1}{4}$, and hence $\lg(r-1) > 1$.

Note that for $r \geq 2$, $r\lg\left(1+\frac{1}{r-1}\right) \geq 1$. For the second term,

$$\begin{aligned}
\left|r\lg\left(1+\frac{1}{r-1}\right) - \tilde{r}\lg\left(1+\frac{1}{\tilde{r}-1}\right)\right| \;&\leq\; \varepsilon r\lg\left(1+\frac{1}{\tilde{r}-1}\right) + r\left|\lg\left(\frac{1+\frac{1}{r-1}}{1+\frac{1}{\tilde{r}-1}}\right)\right| \\
&\leq\; O(\varepsilon)\frac{r}{\tilde{r}-1} + r\left|\lg\left(1+\frac{\frac{\tilde{r}-1}{r-1}-1}{\tilde{r}}\right)\right| \\
&\leq\; O(\varepsilon) + r\,O\left(\frac{1}{\tilde{r}}\left|\frac{\tilde{r}-1}{r-1}-1\right|\right) \\
&\leq\; O(\varepsilon) + O(\varepsilon) \\
&\leq\; O(\varepsilon)r\lg\left(1+\frac{1}{r-1}\right).
\end{aligned}$$

Hence $|g(r) - g(\tilde{r})| \leq O(\varepsilon)g(r)$ as required. □

Theorem 11. *There exists an (ε, δ)-approximation algorithm for H_G using*[2] $O(\varepsilon^{-4}\log^2 n \log^2 \delta^{-1})$ *space.*

Proof. Consider the expectation of the basic estimator:

$$\begin{aligned}
\mathrm{E}[X] \;&=\; \sum_{w\in[n]} \frac{(1\pm O(\varepsilon))d_w}{2|E|} \sum_{r\in[d_w]} \frac{(1\pm O(\varepsilon))}{d_w}(f(r) - f(r-1)) \\
&=\; \frac{1\pm O(\varepsilon)}{2|E|}\sum_{w\in[n]} d_w \lg d_w\,.
\end{aligned}$$

Note that since the graph G is revealed by a random walk, this graph must be connected. Hence $|E| \geq n-1$ and $d_w \geq 1$ for all $w \in V$. But then $\sum_w d_w = 2|E| \geq 2(n-1)$ and therefore,

$$\frac{1}{2|E|}\sum_{w\in[n]} d_w \lg d_w \;\geq\; \lg\frac{2|E|}{n} \;\geq\; \lg 2(1-1/n)\,.$$

The maximum value taken by the basic estimator is,

$$\begin{aligned}
\max[X] \;&\leq\; \max_{1\leq r\leq n}(f(r) - f(r-1)) \\
&\leq\; \left(n\lg\frac{n}{n-1} + \lg(n-1)\right) \\
&\leq\; \left(\frac{n}{n-1} + \lg(n-1)\right) \\
&<\; (2+\lg n)\,.
\end{aligned}$$

[2]Ignoring factors of $\log\log n$ and $\log\varepsilon^{-1}$.

Therefore, by appealing to Lemma 1, we know that if we take c independent copies of this estimator we can get a (ε, δ)-approximation to $\mathrm{E}[X]$ if $c \geq 6\varepsilon^{-2}(2 + \lg n)\ln(2\delta^{-1})/(\lg 2(1 - 1/n))$. Hence with probability $1 - O(\delta)$, the value returned is $(1 \pm O(\varepsilon))H_G$.

The space bound follows because for each of the $O(\varepsilon^{-2} \log n \log \delta^{-1})$ basic estimators we require an ε min-wise hash function using $O(\log n \log \varepsilon^{-1})$ space [18] and a distinct element counter using $O((\varepsilon^{-2} \log \log n + \log n) \log \delta_1^{-1})$ space [4] where $\delta_1^{-1} = O(c\delta^{-1})$. Hence, rescaling ε and δ at the outset gives the required result. □

Our bounds are independent of the length of the stream, m, since there are only n^2 distinct edges, and our algorithms are not affected by multiple copies of the same edge.

Finally, note that our algorithm is actually correct if the multi-set of edges E' arrives in any order, i.e. it is not necessary that (u, v) is followed by (v, w) for some w. Hence our algorithm also fits into the adversarial ordered graph streaming paradigm e.g., [5, 14, 12].

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of ACM Principles of Database Systems*, pages 286–296, 2004.

[3] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*, pages 1–10, 2002.

[5] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[6] L. Bhuvanagiri and S. Ganguly. Estimating entropy over data streams. In *ESA*, 2006.

[7] L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA*, pages 708–713, 2006.

[8] P. Bose, E. Kranakis, P. Morin, and Y. Tang. Bounds for frequency estimation of packet streams. In *SIROCCO*, 2003.

[9] A. Chakrabarti, K. Do Ba, and S. Muthukrishnan. Estimating entropy and entropy norm on data streams. In *STACS*, pages 196–205, 2006.

[10] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *CCC*, pages 107–117, 2003.

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

[12] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.

[14] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[15] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[16] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proc. Internet Measurement Conference*, 2005.

[17] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *SODA*, pages 733–742, 2006.

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

[19] P. Indyk and D. P. Woodruff. Tight lower bounds for the distinct elements problem. In *FOCS*, pages 283–289, 2003.

[20] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

[21] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *ACM SIGMETRICS*, 2006.

[22] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.

[23] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:449–461, 1992.

[24] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast IP networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE)*, 2005.

[25] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, pages 167–175, 2004.

[26] K. Xu, Z. Zhang, and S. Bhattacharya. Profiling internet backbone traffic: Behavior models and applications. In *ACM SIGCOMM*, 2005.