Estimating the Second Frequency Moment

Will Rosenbaum
Updated: May 12, 2013

Department of Mathematics
University of California, Los Angeles

1 Introduction

In this note, we describe a solution to the problem of finding the second frequency moment of a data stream. Suppose we are shown a stream of elements from a finite set S, where we are given elements one by one possibly with repetition. Let $s_1, s_2, \ldots, s_j, \ldots, s_n \in S$ denote the elements we are shown in the order they appear, where we are shown n elements in total. For notational convenience, assume that $S = \{1, 2, \ldots, m\}$ where $m = |S|$. For each $i \in S$, let f_i denote the number of times that i appears in the stream. That is

$$f_i = |\{j \,|\, s_j = i\}| \,. \tag{1}$$

For example, if S is a set of political candidates in an election and we are given a stream of votes ($s_j = i$ means that the j-th person voting votes for candidate i), the number f_i is simply the total number of votes cast for candidate i.

The ***second frequency moment*** of the stream, denoted F_2, is defined to be

$$F_2 = \sum_{i=1}^{m} f_i^2. \tag{2}$$

The second frequency moment is a useful statistic for the stream, and computing or estimating it efficiently is a matter of practical importance. We would like to efficiently compute good estimates for F_2. In particular, we want an algorithm that can process the data quickly, and crucially, use very little memory.

A naive way to estimate F_2 for a stream is just to compute it directly. In particular, for each $i \in S$, we can store a count c_i of the number of times i appears in the stream. Starting with $c_i = 0$ for all i, each time we see a new element in stream, we increment the counter corresponding to that element. Then when we reach the end of the stream, we simply compute $F_2 = c_1^2 + c_2^2 + \cdots + c_m^2$. In order to implement this procedure, we need store the numbers $c_1, c_2, \ldots, c_m$, where a given c_i can be as large as n. Thus we need $\Theta(m \log n)$ bits of memory to compute F_2 naively. This is fine if m is small (relative to n, the number of elements in the stream) but what if $m = |S|$ is large? For example, S could be a set of words in a dictionary, or the set of people on Facebook, or the set of IP addresses of computers. Then this algorithm is no longer efficient, and we have to be more clever.

It turns out that computing F_2 precisely and deterministically, even if we are infinitely clever in designing our algorithms, requires $\Omega(n)$ space for a worst-case input. Therefore we must be satisfied to approximate F_2 if we want to work efficiently. Here, we will describe an algorithm that will approximate F_2 using $\mathcal{O}(\log n)$ space.

2 The Algorithm

The algorithm works in the following way: let $e_1, e_2, \ldots, e_m$ be iid random variables with $\mathbf{P}(e_i = 1) = \mathbf{P}(e_i = -1) = \frac{1}{2}$. Define

$$y = \sum_{i=1}^{m} e_i f_i. \tag{3}$$

Geometrically, y is a random (linear) projection of the vector $\mathbf{f} = (f_1, f_2, \ldots, f_m) \in \mathbf{R}^m$ to $\mathbf{R}$. It is crucial that we can compute y without having to store the entire vector $\mathbf{f}$. To accomplish this, before seeing the stream, we initialize $y = 0$. As elements in the stream appear, we update y: if we see $i \in S$ in the stream, we update $y \mapsto y + e_i$. Thus we can maintain y without having to store the array $\mathbf{f}$ as in the naive approach described in the introduction.

∗ k will be chosen later depending on the desired accuracy of the approximation

We repeat this procedure k times* independently (independent choices of the e_i) in parallel and get an array of numbers $\mathbf{y} = (y_1, y_2, \ldots, y_k) \in \mathbf{R}^k$. That is, we choose iid ± 1–random variables e_{ji} for $j = 1, 2, \ldots, k$ and $i = 1, 2, \ldots, m$. $\mathbf{y}$ is the projection given by

$$\mathbf{y} = M\mathbf{f} \quad \text{where the } ji\text{–entry of } M \text{ is } e_{ji}. \tag{4}$$

The vector $\mathbf{y}$ is known as a ***sketch*** of the frequency vector $\mathbf{f}$. In general, $k \ll m$ so that maintaining $\mathbf{y}$ is much less memory intensive than maintaining $\mathbf{f}$.

Now we must show that we can compute a reasonable estimate of F_2 from the sketch $\mathbf{y}$. To this end, we define the estimator

$$\widehat{F_2} = \frac{1}{k} \sum_{j=1}^{k} y_j^2 \tag{5}$$

In order to show that $\widehat{F_2}$ is a good estimator for F_2, we will show that $\mathbf{E}(\widehat{F_2}) = F_2$ and we will bound the variance of each y_j^2. Together, these will show that for modest-sized k, the probability that $\widehat{F_2}$ is far from F_2 is small.

Proposition 1.

$$\mathbf{E}(\widehat{F_2}) = F_2. \tag{6}$$

Proof. By linearity of expectation, it suffices to show that $\mathbf{E}(y^2) = F_2$ for a single y. We compute

$$\begin{aligned} \mathbf{E}(y^2) &= \mathbf{E}\left(\sum_{i=1}^{m} e_i f_i\right)^2 \\ &= \sum_i \mathbf{E}(e_i^2) f_i^2 + 2 \sum_{i<j} \mathbf{E}(e_i)\,\mathbf{E}(e_j) f_i f_j \\ &= F_2. \end{aligned}$$

The second equality holds because for $i \neq j$, e_i and e_j are independent, while the final equality holds because $e_i^2 = 1$ and $\mathbf{E}(e_i) = 0$ for all i. □

Proposition 2.

$$\operatorname{var}(y^2) \leq 2L_2^2. \tag{7}$$

Proof. First we compute

$$\begin{aligned}\mathbf{E}(y^4) &= \mathbf{E}\left(\sum_{ijk\ell} e_i f_i e_j f_j e_k f_k e_\ell f_\ell\right) \\ &= \sum_{ijk\ell} f_i f_j f_k f_\ell \, \mathbf{E}(e_i e_j e_k e_\ell) \\ &= \sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2.\end{aligned}$$

The final equality holds by independence because the terms with odd powers of e_i evaluate to zero, while there are $\binom{4}{2} = 6$ terms of the form $f_i^2 f_j^2$. Now we can compute the variance

$$\begin{aligned}\mathrm{var}(y^2) &= \mathbf{E}(y^4) - \mathbf{E}(y^2)^2 \\ &= \sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2 - \sum_i f_i^4 - 2\sum_{i<j} f_i^2 f_j^2 \\ &= 4\sum_{i<j} f_i^2 f_j^2 \\ &\le 2\left(\sum_i f_i^2\right)^2 \\ &= 2L_2^2.\end{aligned}$$

This is the desired result. □

By the definition of $\widehat{F}_2$ (equation (5)), we immediately get

$$\mathrm{var}(\widehat{F}_2) = \frac{2}{k} F_2^2. \tag{8}$$

Applying Chebyshev's inequality $\widehat{F}_2$ gives

$$\mathbf{P}\left(\left|\widehat{F}_2 - F_2\right| \ge c\sqrt{\frac{2}{k}} F_2\right) \le \frac{1}{c^2}. \tag{9}$$

By choosing $k = \mathcal{O}(1/\varepsilon^2)$, we obtain a $(1 \pm \varepsilon)$–approximation with constant probability.

3 Technical Details

The sketching algorithm described in the previous section certainly does give a good approximation of F_2. However, it does not achieve the desired space efficiency. In order to implement the algorithm as stated, we must store the matrix M, which contains km entries from $\{\pm 1\}$. This is unacceptable.

To avoid storing M explicitly, we will construct an explicit set S of $\mathcal{O}(m^2)$ ± 1–vectors that is 4–***wise independent***. That is, for every four distinct indices $1 \le i_1 < i_2 < i_3 < i_4 \le m$ and $e_1, e_2, e_3, e_4 \in \{\pm 1\}$, exactly $1/16$ fraction of the $v \in S$ satisfy $v_{i_j} = e_j$ for $j = 1, 2, 3, 4$. Then we can define M by choosing k vectors uniformly at random from S to be the rows of M. This requires only $\mathcal{O}(k \log m)$ bits of memory as $|S| = \mathcal{O}(m^2)$.* In

* We will not store the set S in memory, but will instead be able to compute the entries of a random $v \in S$ from it's "address" (i.e. a random string of $\mathcal{O}(\log m)$ bits) on the fly.

the analysis of the algorithm in the previous section, we only used the fact that the e_i were 4–wise independent. Thus, choosing the rows of M uniformly at random from S will give the same performance guarantees of the algorithm while improving the space complexity exponentially!

Theorem 3. Suppose $m' = 2^k - 1$ and $d = 2t + 1$. Then given a representation of the finite field $\mathbf{F} = \mathbf{F}_{2^k}$ of size 2^k there exists a symmetric probability Ω space of size $2(m'+1)^t$ and d-wise independent random variables $e_1, \ldots, e_m$ over Ω which take on the values 0 and 1 with probability $\frac{1}{2}$.

Taking $t = 2$ and $k = \lceil \log_2(m+1) \rceil$ gives us the set of $\mathcal{O}(m^2)$ 4–wise independent vectors we need to implement the algorithm efficiently. To give an explicit representation of the field $\mathbf{F}$, it suffices to give an irreducible polynomial of degree k over $\mathbf{F}_2$. For such a polynomial p, $\mathbf{F} \simeq \mathbf{F}_2[x]/p$. Therefore, we can represent elements of $\mathbf{F}$ as binary strings of length k and perform all arithmetic modulo p. I assume that such a p can be efficiently. In order to prove the theorem, we will need the following lemma.

Lemma 4. Let $x_1, x_2, \ldots, x_{m'}$ denote the nonzero elements of $\mathbf{F}$, represented as a column vector of length k over $\mathbf{F}_2$. Let H be the $(1 + kt) \times m'$ matrix defined by

$$H = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{m'} \\ x_1^3 & x_2^3 & \cdots & x_{m'}^3 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{2t-1} & x_2^{2t-1} & \cdots & x_{m'}^{2t-1} \end{pmatrix}. \tag{10}$$

Then any set of $d = 2t + 1$ columns of H is linearly independent over $\mathbf{F}_2$.

Proof. Suppose $J \subset \{1, 2, \ldots, m'\}$ has size $|J| = 2t+1$ and the numbers $z_j \in \mathbf{F}_2$ $(j \in J)$ satisfy

$$\sum_{j \in J} z_j H_j = 0 \tag{11}$$

where H_j is the j–th column of H. We must show that $z_j = 0$ for all j. Notice that (11) implies that

$$\sum_{j \in J} z_j x_j^i = 0 \tag{12}$$

for $i = 0$ and $i = 1, 3, \ldots, 2t - 1$. We will show that that in fact (12) holds for all $i = 0, 1, 2, \ldots, 2t$. To see this, suppose $i = 2^\ell a$ where $i \in \{0, 1, \ldots, 2t\}$ and a is odd. Then we have

$$0 = \left(\sum_{j \in J} z_j x_j^a \right)^{2^\ell} = \sum_{j \in J} z_j^{2^\ell} x_j^{2^\ell a} = \sum_{j \in J} z_j x_j^i.$$

The first equality is by (12), the second holds because $(x + y)^2 = x^2 + y^2$ over a field of characteristic 2, while the final equality holds because $z_j \in \mathbf{F}_2$. Therefore, (12) holds for all

$i \in \{0, \ldots, 2t\}$. The $d \times d$ matrix V over $\mathbf{F}$

$$V = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_{j_1} & x_{j_2} & \cdots & x_{j_d} \\ x_{j_1}^2 & x_{j_2}^2 & \cdots & x_{j_d}^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_{j_1}^d & x_{j_2}^d & \cdots & x_{j_d}^d \end{pmatrix}$$

is a Vandermonde matrix, which has determinant $\prod_{j<j'}(x_j - x_{j'}) \neq 0$, as $x_j \neq x_{j'}$ for $j \neq j'$. Therefore, since V is invertible, the homogeneous linear system of equations (12) has only the trivial solution. That is, $z_j = 0$ for all $j \in J$. □

Proof of theorem 3. Define $\Omega = 2^{\{1,2,\ldots,1+kt\}}$, the set of subsets of $\{1, 2, \ldots, 1 + kt\}$, endowed with the uniform probability measure. Notice that $|\Omega| = 2^{kt+1} = 2(m' + 1)^t$ as promised. We can view elements of Ω as binary strings of length $1 + kt$. That is $\omega = (\omega_1, \omega_2, \ldots, \omega_{1+kt}) \in \Omega$ where $\omega_i = 1$ if and only if i is contained in subset of $\{1, \ldots, 1 + kt\}$ represented by ω and $\omega_i = 0$ otherwise. Given $\omega \in \Omega$, we define

$$e_j = \sum_{i=1}^{1+kt} \omega_i h_{ij}$$

where h_{ij} is the ij–entry of H as in Lemma 4.

It remains to be shown that the e_j are d–wise independent and that each takes the values 0 and 1 with probability $1/2$. To this end, let $J \subset \{1, \ldots, m'\}$ with $|J| \leq d$. We must show that given any $b \in \{0, 1\}^{|J|}$, there are exactly $2^{1+kt-|J|}$ vectors ω such that

$$\sum_{i=1}^{1+kt} \omega_i h_{ij} = b_j \quad \text{for all} \quad j \in J. \tag{13}$$

By the lemma, the columns H_j are independent for $j \in J$, hence the equations (13) are independent. Therefore, the set of solutions to (13) is an affine subspace of $\mathbf{F}_2^{1+kt}$ of dimension $1 + kt - |J|$. In particular, there are $2^{1+kt-|J|}$ solutions to (13) as desired. □

To summarize, the algorithm works as follows. We choose and store a $(1+kt)$-bit string ω uniformly at random, where $1 + kt = \mathcal{O}(\log m)$ ($t = 2$ is a fixed constant), as well as an irreducible polynomial of degree $k = \mathcal{O}(\log m)$ over $\mathbf{F}_2$. We maintain the integer y, where y is initialized to be 0. When we see an element j in the stream, we compute $e_j = \sum_i \omega_i h_{ij}$ over $\mathbf{F}_2$. The elements h_{ij} need not be stored, so long as we fix some order on the elements of $\mathbf{F}$ relative to p, say lexicographical order, so that x_j can be easily computed from j. Then we compute the $\mathcal{O}(1)$ elements $1, x_j, x_j^3, \ldots, x_j^{2t-1}$ (each of which has size $\mathcal{O}(\log m)$) from which we can compute the e_j. We then take $y \mapsto y + 2e_j - 1$. With the exception of storing y, all of these operations require $\mathcal{O}(\log m)$ bit of memory, while y requires $\mathcal{O}(\log n)$ bits. Thus, the entire algorithm requires $\mathcal{O}(\log m + \log n)$ bits of memory.

By running this algorithm $\mathcal{O}(1/\varepsilon^2)$ times independently (in the choice of ω) and in parallel, we obtain the following:

Theorem 5. Given a data stream of length n containing elements from a set of size m and and given $\varepsilon > 0$, there exists a streaming algorithm that with constant probability computes a $1 \pm e$ approximation $\widehat{F}_2$ of the second frequency moment F_2 using $\mathcal{O}((\log m + \log n)/\varepsilon^2)$ bits of memory.

4 Extension to Other Frequency Moments

In general for $p \in [0, \infty)$ we define the p-***th frequency moment*** of a stream to be

$$F_p = \sum_{i=1}^{m} f_i^p \tag{14}$$

where as before $f_i = |\{j \mid s_j = i\}|$. We also define

$$F_\infty^* = \sup_i \{f_i\} \tag{15}$$

Writing $\mathbf{f} = (f_1, f_2, \ldots, f_m) \in \mathbf{R}^m$, the quantities F_p are related to the ℓ^p norm of $\mathbf{f}$:

$$F_p = \|\mathbf{f}\|_p^p \quad \text{and} \quad F_\infty^* = \|\mathbf{f}\|_\infty . \tag{16}$$

In this setting, we can identify a class of potential streaming algorithms as methods for "dimension reduction." For example, the algorithm detailed for F_2 can be viewed as describing a (random) linear map $A_2 : \mathbf{R}^m \longrightarrow \mathbf{R}^k$. The efficiency and correctness of the algorithm are due to the facts that $k \ll m$ and that an approximation for F_2 can be computed from the image $A(\mathbf{f})$. With this view of the problem, one is led to ask if it is always possible to find a linear* map $A_p : \mathbf{R}^m \longrightarrow \mathbf{R}^k$ for some $k \ll m$ such that $\|\mathbf{f}\|_p$ can be approximated from $A_p(\mathbf{f})$. A particularly nice property we could hope for would be if

$$\|\mathbf{f}\|_p \approx \|A_p(\mathbf{f})\|_p .$$

If such functions A_p exist, we would be well on our way to finding efficient algorithms for approximating F_p. Given such a function, we would just need to show that we can give a short description of the function (i.e., a description that is logarithmic, say, in the dimension m). It turns out that such functions do exist for $p \in (0, 2]$. For $p > 2$, however, algorithms for computing F_p require $n^{\Omega(1)}$ memory.

∗ Linearity is necessary in order to process updates to $\mathbf{f}$ as elements appear in the stream without seeing all of $\mathbf{f}$ at once.

4.1 F_p for $p \in (0, 2]$

Definition 6. Let D be a probability distribution and $x_1, \ldots, x_n, x \sim D$ iid random variables. We say that D is p-***stable*** if for any $a = (a_1, \ldots, a_n) \in \mathbf{R}^n$ we have

$$a_1 x_1 + \cdots + a_n x_n \sim \|a\|_p \, x.$$

That is, the linear combination $a_1 x_1 + \cdots + a_n x_n$ has the same distribution as $\|a\|_p \, x$.

Example 7. The Guassian distribution $N(0, 1)$ is 2-stable. To see this, we recall some properties of the normal distribution. Suppose x and y are iid Gaussian random variables and $a \in R$. Then

1. $x + y$ and ax are Gaussian;
2. $var(ax) = a^2 \operatorname{var}(x)$;
3. $\operatorname{var}(x + y) = \operatorname{var}(x) + \operatorname{var}(y)$.

From these properties (and the fact that a Gaussian distribution is uniquely determined by its expectation and variance), it is clear that for iid $N(0, 1)$ random variables $x_1, \ldots, x_n$ and x, and $a = (a_1, \ldots, a_n) \in \mathbf{R}^n$

$$a_1 x_1 + \cdots + a_n x_n \sim \|a\|_2 \, x.$$

Using the fact that $N(0,1)$ is 2-stable, we can construct another candidate algorithm for approximating F_2. Let $x_1, \ldots, x_m$ be independent $N(0,1)$ random variables. For a frequency vector $\mathbf{f} = (f_1, \ldots, f_m)$, define

$$y = \sum_{i=1}^{m} f_i x_i.$$

Since $N(0,1)$ is 2-stable, $y \sim N(0, \|\mathbf{f}\|_2^2)$. Therefore, an estimate of the variance of y will give an estimate of F_2. To perform such an estimate, we repeat this process k (k to be determined later) times in parallel. For $j = 1, 2, \ldots, k$ define

$$y_j = \sum_{i=1}^{m} f_i x_{ji} \quad \text{where} \quad x_{ji} \text{ are independent, } N(0,1).$$

Then the $y_j \sim N(0, \|\mathbf{f}\|_2^2)$ are independent, so we can can estimate the variance F_2 to be

$$\widehat{F_2} = \frac{1}{k} \sum_{j=1}^{k} y_j^2.$$

Notice that $\mathbf{E}(\widehat{F_2}) = F_2$ because $\mathbf{E}(x_{ji} x_{ji'}) = 0$ for $i \neq i'$ by independence, and $\mathbf{E}(x_{ji}^2) = 1$.

Fact 8. There exist p-stable distributions for $p \in (0, 2]$.

Using this fact, we can estimate F_p in the following manner: generate random variables x_{ji} according to a p-stable distribution. As before, we form the variables $y_j = \sum f_i x_{ji}$. The estimator $\widehat{F_p}$ is computed from the y_j, although this computation is not as straightforward as the $p = 2$ case. We must instead analyze quantiles of the p-stable distribution used. Such an analysis allows us to infer the value of F_p from the y_j given that the y_j have the same distribution as $F_p^{1/p} x$ where x has the same distribution as the x_{ji}.

In order to implement these ideas as efficient algorithms, we still have a few problems to solve. First, the p-stable distributions are continuum-valued random variables, which must be approximated by some discretization. Second, we cannot store the matrix M as doing so requires $\Omega(km)$ storage. So we must show that we can choose random elements from some small probability space, analogously to the method described in section 3. Both of these problems can be overcome, but a description of the methods involved is outside of the scope of this essay.

4.2 F_p **for** $p > 2$ For the case where $p > 2$, we will give an algorithm that gives a $(1 \pm \varepsilon)$-approximation of F_p using $\tilde{\mathcal{O}}(pm^{1-1/p}/\varepsilon^2)$ bits of memory. For large p, this algorithm is close to the best possible. A theoretical lower bound of $\Omega(m^{1-2/p}/(1+\varepsilon)^2)$ can be obtained from communication complexity, and this bound can be achieved although the optimal algorithm is out of reach for this short essay.

The idea behind the algorithm we give for estimating F_p is to use sampling. For $p > 2$, the numbers f_i that contribute most to F_p appear as the most frequent elements in the stream. Sampling a random i_j in the stream and computing f_{i_j} should therefore allow us to approximate F_p, as the most frequent elements in the stream are most likely to be sampled. However, for large p, F_p is highly dependent on the largest f_i, which may be as small as n/m. Therefore, as p becomes very large we expect to need to sample a large portion of

the stream in order for this method to give accurate results. This intuition is correct, as the following theorem and lower-bound result demonstrate.

In order to describe an algorithm for F_p and $p > 2$, first suppose that our algorithm read the stream $s_1, s_2, \ldots, s_n$ twice in the same order. On the first pass, we pick an index $j \in \{1, \ldots, n\}$ uniformly at random and store the value $i_j \in \{1, \ldots, m\}$. On the second pass, we compute f_{i_j} by simply counting the incidences of i_j, and return the value $y' = nf_{i_j}^{p-1}$. Notice that

$$\mathbf{E}(y') = \sum_i \frac{f_i}{n} n f_i^{p^{i-1}} = \sum_i f_i^p = F_p.$$

Furthermore, the variance of y can be bounded using the computation

$$\mathbf{E}(y'^2) = \sum_i \frac{f_i}{n} n^2 f_i^{2p-2} = n \sum_i f_i^{2p-2} = nF_{2p-1} \leq m^{1-1/p} F_p^2.$$

The final inequality is nontrivial and is proved in [1]. Repeating this process $\mathcal{O}(m^{1-1/p})$ times independently in parallel and taking the mean of the results, we obtain an algorithm for approximating F_p with constant probability (apply Chebyshev's inequality to prove this bound).

In order to achieve a similar result in one pass, again pick j uniformly at random from $\{1, \ldots, n\}$ and compute r, the number of occurrences of i_j appearing after j in the stream. We take as our estimator

$$y = n(r^p - (r-1)^p). \tag{17}$$

Again, we compute the expected value of y:

$$\mathbf{E}(y) = n\,\mathbf{E}(r^p - (r-1)^p) = n\frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{f_i}(j^p - (j-1)^p) = \|\mathbf{f}\|_p^p$$

where the final equality holds by the telescoping sum. Notice that $y \leq npr^{p-1} \leq py'$. Thus

$$\operatorname{var}(y) \leq \mathbf{E}(y^2) \leq p^2\,\mathbf{E}(y'^2) \leq p^2 m^{1-1/p} F_p^2.$$

Taking $k = C_p m^{1-1/p} F_p^2/\varepsilon^2$ independent estimators $y_1, \ldots, y_k$, we define

$$z = \frac{1}{k}\sum_j y_j.$$

Applying Chebyshev's inequality now gives

$$\mathbf{P}(|z - F_p| \geq \varepsilon F_p) \leq 1/4.$$

Thus, we get a one-pass algorithm using $\tilde{\mathcal{O}}(pm^{1-1/p}/\varepsilon^2)$ space for the approximation.

In order to prove a lower bound for the space requirement for computing F_p, we require a result that relies heavily on communication complexity.

Fact 9. Any single-pass streaming algorithm that gives a c–approximation of $\|\mathbf{f}\|_\infty = F_\infty^*$ requires space $\Omega(m/c^2)$.

Granting this fact, we can prove a lower bound on the efficiency of streaming algorithms for estimating L_p for $p > 2$.

Theorem 10. Any single-pass streaming algorithm that gives a c–approximation of F_p requires space $\Omega(m^{1-2/m}/c^2)$ space.

Proof. We will show that any c–approximation of $\|\mathbf{f}\|_p$ gives a $cn^{1/p}$–approximation of $\|\mathbf{f}\|_\infty$. To see this, suppose y is such an approximation. That is,

$$\begin{aligned}
\frac{1}{c}\|\mathbf{f}\|_p \leq y \leq c\|\mathbf{f}\|_p &\implies \frac{1}{c}\left(\sum_{i=1}^m f_i^p\right)^{1/p} \leq y \leq c\left(\sum_{i=1}^m f_i^p\right)^{1/p} \\
&\implies \frac{1}{cn^{1/p}}\left(n\sum_{i=1}^m f_i^p\right)^{1/p} \leq y \leq cn^{1/p}\left(\frac{1}{n}\sum_{i=1}^m f_i^p\right)^{1/p} \\
&\implies \frac{1}{cn^{1/p}}\|\mathbf{f}\|_\infty \leq y \leq cn^{1/p}\|\mathbf{f}\|.
\end{aligned}$$

The theorem now easily follows from Fact 9. □

This lower bound shows that the approximation algorithm described for F_p, $p > 2$ is near optimal. It turns out that the given lower bound is correct and a space-optimal algorithm has been devised, although a description of the optimal algorithm is beyond the scope of this essay.

5 Notes and Sources

The seminal paper on streaming algorithms is [1], which contains the algorithm for F_2 described in the first two sections of this essay, as well as the algorithm for F_p, $p > 2$. Theorem 3 and its proof are from the chapter on "derandomization" in [2].

The algorithm for efficiently computing F_p for $p \in (0, 2]$ is due to Indyk [3]. Much more on stable distributions can be found in [7].

All of the proofs for lower bounds given here rely on communication complexity. Work in this field was initiated by Yao [6]. A very thorough reference for communication complexity is [5].

The presentation given here was strongly influenced by [4]. This reference contains lecture notes for a course taught by Indyk on streaming algorithms and offers an extremely good overview of other topics in this exciting field.

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

[2] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.

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

[4] Piotr Indyk. Sketching, streaming and sub-linear space algorithms. http://stellar.mit.edu/S/course/6/fa07/6.895/, 2007.

[5] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[6] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.

[7] V. M. Zolotarev. *One-dimensional Stable Distributions*, volume 65. American Mathematical Society, translations of mathematical monographs edition, 1986.