COMS-E6998: Information Theory in Computer Science **Fall 2017**

Lecture 8: Streaming

Lecturer: Omri Weinstein *Scribes: scribe-name1,2,3*

8.1 Motivation

We are all used to thinking of linear-time algorithms as being very efficient (in fact, the best possible, since we need to look on all the input to compute a general function of it). However, many real-life applications (e.g., monitoring network packet traffic through routers, analysis of genomic sequencing data, eCommerce transactions etc.) involve data in volumes and speed that is far too large to store on a centralized machine. As an example, think of Amazon or Walmart transactions – while storing the entire *online* update stream on one machine is infeasible, the company would still like to keep some statistical estimates of the transactions (for example, what is the median age of customers that bought products in category X,X, or Z)?

Formal model: In the *general streaming model*[1], we have an *online* stream $X = x_1, \ldots, x_n$ of n elements $x_i \in [m]$ (n is not a-priori known, and should be thought as rather huge, say, 1EB). The goal is to process this stream in *on-the-fly* (i.e., make a *single* pass over the stream), and compute (or more modestly, approximate) a function of the final stream

$$g(x_1, \ldots, x_n)$$

using minimum storage space, ideally constant or $poly(\lg n, \lg m)$ bits, and in any case, *sublinear* in n.

Definition 8.1.1. *We say that a (1-pass) streaming algorithm* (ε, δ)-approximates $g(X)$ *using s bits of space, if there's a randomized online algorithm that uses s space and returns an estimate v s.t*

$$\Pr[v \in (1 \pm \varepsilon) g(X)] \geq 1 - \delta$$

for any stream $X = x_1, \ldots, x_n$.

8.2 Frequency Moments

The canonical problem in the field of data stream algorithms is computing the "frequency moments" of a stream. This problem was studied in the paper that kickstarted the field (Alon et al. [AMS99]), and the data stream algorithms has been obsessed with it ever since.

For a data stream $X = x_1, \ldots, x_n$, the kth frequency moment of X is

$$F_p(X) = \begin{cases} |\{j : f_j > 0\}| & \text{if } p = 0 \\ max_j |f_j| & \text{if } = \infty \\ \|f\|_p^p = \sum_{j=1}^m f_j^p & \text{otherwise} \end{cases} \tag{8.1}$$

where $f_j := |\{i : x_i = j\}|$ is the frequency of element $j \in [m]$.

[1] **Remark:** In the so-called *Turnstile streaming model*, x_i's are simply increments $x_i \in (j, \delta_j)$ to a specific coordinate $j \in [m]$.

F_0 is the number of distinct elements in the stream while F_1 is the number of elements (with repetition). Clearly, $F_1 = n$ and can be computed (exactly) using $O(\lg n)$ space (we'll soon see that we can do exponentially better if we allow small approximation error). F_k is the largest frequency, and its easy to imagine wanting to know this (for example to detect a denial-of-service attack at a network switch, or detecting the most popular product on Amazon yesterday). It is easy to see that F_0 can be computed exactly using m bits of space. In fact, all the moments can be computed using $m \lg n$ space by keeping track of the frequency vector X.

Intuitively, it might appear difficult to improve over the trivial solution. For F_0, for example, it seems like you have to know which elements you've already seen (to avoid double-counting them), and theres an exponential (in m) number of different possibilities for what you've seen in the past. In light of this , the following result of Alon, Matias and Szegedy is quite surprising:

Theorem 8.2.1 (F_2 estimation, AMS)**.** *There is a (randomized) streaming algorithm that (ε, δ)-approximates F_2 (and F_0 = number of distinct elements), using space*

$$s = O(\varepsilon^{-1} \lg(1/\delta) \cdot (\lg n + \lg m)).$$

The following general result (incorporating subsequent improvements) summarizes the space complexity of computing various frequency moments:

Theorem 8.2.2. *The space complexity of approximating frequency moments is as follows:*

1. *For $P \in [0, 2]$ there is a randomized streaming algorithm that ε, δ-approximates F_p in space $s = O(poly(\lg m, \lg n))$.*
2. *For $p > 2$, any randomized streaming algorithm that (ε, δ)-approximates F_p, requires space $s = \Theta(m^{1-2/p})$.*

In particular, maintaining a constant approximation to F_∞ requires $\Omega(m)$ space, which is the main result we'll prove next lecture.

8.2.1 General Approach to Streaming Algorithms

As we mentioned before, achieving *sublinear* space (in m, n) entails resorting to *approximation.* The following key idea explains our general approach to streaming problems (in particular, to momet estimation):

Key idea: Find a *succinct*, dynamically-computable unbiased estimator A for g :

$$\mathbb{E}[A(X)] \approx g((X).$$

Concentration : Expectation is typically not enough as we'd like to output a close answer w.h.p.

One natural way to achieve concentration is to maintain multiple (parallel) independent estimators, and take their average, since averaging reduces variance (recall that $Var[X] := \mathbb{E}[(X - \mathbb{E}[X])^2]$):

$$Var\left[\frac{1}{t}\sum_{i=1}^{t} A_i\right] = \frac{Var[A]}{t}.$$

It turns out that the desirable property of our estimator is bounded variance in terms of its expectation, i.e.,

$$Var[A] \leq O(\mathbb{E}^2[A]).$$

We can then use Chebychev's inequality to argue that $O_{\delta,\varepsilon}(1)$ number of repetitions are enough to get a (ε,δ)-approximation to the desired function $g(X)$:

Lemma 8.2.3. *Suppose $E[A] = g(X)$ and further $Var[A] \leq K \cdot \mathbb{E}^2[A]$. Then for $t = O(K\varepsilon^{-2} \lg 1/\delta)$,*

$$\Pr_{A_1,\ldots,A_t}\left[\frac{1}{t}\sum_{i=1}^{t} A_i \in (1 \pm \varepsilon)g(X)\right] \geq 1 - \delta.$$

Proof idea: We apply Chebychev's inequality ($\Pr[|Y - \mathbb{E}[Y]| > c \cdot \sigma(Y)] \leq 1/c^2$), with $c := \varepsilon g(X)$. To this end, let us denote $Z := \frac{1}{t}\sum_{i=1}^{t} A_i$. By our assumptions,

$$Var[A] \leq K\mathbb{E}^2[A] = K\mathbb{E}^2[g(x)] \implies \sigma(A) \leq \sqrt{K}g(X) \implies \sigma(Z) \leq \sqrt{\frac{K}{t}}g(X).$$

$$\Pr_Z[Z \notin (1 \pm \varepsilon)g(X)] = \Pr_Z[|Z - \mathbb{E}[Z]| > \varepsilon g(X)] \leq \Pr_Z\left[|Z - \mathbb{E}[Z]| > \varepsilon\sqrt{\frac{t}{K}}\,\sigma(Z)\right] \leq \frac{K}{\varepsilon^2 t} \quad \text{(Chebychev)}, \tag{8.2}$$

so setting $t := \frac{K}{\varepsilon^2\delta}$ makes this probability $\leq \delta$.

Remark 8.2.4. *Using a mean-of-medians trick, can get the dependence on δ to logarithmic ($\lg(1/\delta)$).*

8.2.2 F_1: Estimating the Length of a Stream (Morris' Algorithm [Morris78])

Suppose we're given a stream of characters, and we'd like to simply estimate the length of the stream n. This can be done trivially using a $(\lg n)$-bit counter, hence requires $s = \Theta(\lg n)$ space naively. If we're willing to settle for (arbitrarily small constant) *approximation*, then turns out we can do much better – using only $s = O(\lg \lg n)$ space:

A streaming algorithm for estimating F_1
1. Initialize a counter C to 0.
2. For each incoming character, set $C \longleftarrow C + 1$ w.p $1/2^C$.
3. Output $A := 2^C - 1$.

Figure 8.1: A communication protocol.

Intuitively, the streaming algorithm A is attempting to store the *logarithm* of n. Indeed, by induction on n, it is straightforward to prove that

$$\mathbb{E}[2^{C_n}] = n + 1,$$

The harder step is showing that $Var[A] = O((\mathbb{E}[A])^2)$:

A streaming algorithm for estimating F_2 ("Tug of War")

1. Initialize:
2. Choose $h : [m] \mapsto \{\pm 1\}$ independently at random[a] (e.g., (+1,+1,-1,+1,-1,-1,-1,+1,. . . , -1))
3. $Y \longleftarrow 0$.
4. Process:
5. foreach i do
6. $y \longleftarrow y + h(x_i)$
7. Output: Y^2.

[a] We'll actually see that choosing $h()$ randomly from a family of 4-wise independent hash functions suffices, and requires only $O(\lg m)$ bits of space to represent.

Figure 8.2: A communication protocol.

8.2.3 F_2: The [AMS] Algorithm

As stated, the space complexity of the algorithm (A) is $O(n)$ – we'll take care of this soon, but let's first see the analysis:

$$\mathbb{E}[A] = \mathbb{E}[Y^2] = \mathbb{E}\left(\sum_{i=1}^{n} x_j \cdot h(x_i)\right)^2 = \mathbb{E}\left(\sum_{j=1}^{m} f_j \cdot h_j\right)^2$$
$$= \mathbb{E}\left(\sum_{j=1}^{m} f_j \cdot h_j\right) \cdot \left(\sum_{j=1}^{m} f_j \cdot h_j\right) = \mathbb{E}\sum_{j=1}^{m} f_j h_j^2 + \sum_{i \neq j} f_i f_j h_i h_j$$
$$= \sum_{j=1}^{m} f_j^2 + \sum_{i \neq j} f_i f_j \mathbb{E}[h_i h_j] = \sum_{j=1}^{m} f_j = F_2(X).$$

For the variance calculation, we compute $\mathbb{E}[A^2] = \mathbb{E}[Y^4]$:

$$\mathbb{E}[Y^4] = \mathbb{E}\left(\sum_{i=1}^{n} x_j \cdot h(x_i)\right)^4 = \mathbb{E}\sum_i \sum_j \sum_k \sum_\ell f_1 f_j f_k f_\ell h_i h_j h_k h_\ell \tag{8.3}$$
$$= \sum_i \sum_j \sum_k \sum_\ell f_1 f_j f_k f_\ell \cdot \mathbb{E}[h_i h_j h_k h_\ell] \tag{8.4}$$

Now, if one out of i, j, k, ℓ occurs only *once* in the above summand, the expectation is 0. Therefore the only terms that contribute are when all 4 are equal and when they form 2 pairs. There are 3 ways that they can form 2 pairs, depending on which of j, k, ℓ the value i is paired with. Moreover, in these cases $\mathbb{E}[h_i h_j h_k h_\ell] = 1$. Therefore we can continue top write

$$\mathbb{E}[Y^4] = \sum_{j=1}^{m} f_j^4 + 3\sum_{i\neq j} f_i^2 f_j^2 = \sum_{j=1}^{m} f_j^4 + 3\left(\sum_{i,j\in[m]} f_i^2 f_j^2 - \sum_{j=1}^{m} f_j^4\right)$$
$$= F_4 + 3(F_2^2 - F_4) = 3F_2^2 - 2F_4.$$

So,

$$Var[A] = Var[Y^2] = \mathbb{E}[Y^4] - \mathbb{E}^4[Y] = 3F_2^2 - 2F_4 - F_2^2 < 2F_2^2 = 2\mathbb{E}^2[A].$$

So we have the desired guarantees. The only missing observation is that we only needed independence of the h_j's to argue that the expectation of a product of ≤ 4 terms = product of their expectations. But this is true even if we chose the h_j's to be only *4-wise independent*: Indeed, the any 4 columns of the hash table is uniform random, hence the corresponding expectation is the product of expectations. Fortunately (and crucially!), 4-wise independent hash functions can be represented using only $O(\lg m)$ bits of space! Hence we get the desired space if we replace step 2 in the algorithm with a random 4-wise independent hash function, so we can repeat algorithm A $t = 2/\delta\varepsilon^2$ times in parallel and get the desired accuracy.

Discussion: Connection to JL dimension reduction. Linear sketches for compressing the data..

8.3 Lower Bounds via Communication Complexity

Reduction to 1-way communication complexity. There is a very elegant connection b/w streaming and communication complexity: Suppose we partition the stream into past and future sub streams, hand Alice the "past" and Bob the "future", and ask then to compute $f(X_A, X_B)$. If there was an s-space (rand) streaming algorithm for f, then it induces a *1-way* (rand) CC protocol with s bits of CC.

Proposition 8.3.1. *Any (randomized) streaming algorithm that computes F_∞ exactly requires $s = \Omega(m)$ space.*

Proof idea: Reduction from (randomized) Set Disjointness.

8.3.1 A Lower Bound for Approximating F_∞

Theorem 8.3.2. *Any (randomized) streaming algorithm that c-approximates F_∞ requires space $s = \Omega(m/c^2)$.*

Corollary 8.3.3. *Any streaming algorithm that c-approximates F_p (with constant probability) requires $s = \Omega(m^{1-2/p})$ space.*

Proof: $\|X\|_\infty \leq \|X\|_p \leq m^{1/p}\|X\|_\infty$, so $\|X\|_p$ is an $m^{1/p}$-approximation for $\|X\|_\infty$.

References