CSE 522: Sublinear (and Streaming) Algorithms **Spring 2014**

Lecture 8: Pseudorandomness and Sketching

April 23, 2014

Lecturer: Paul Beame *Scribe: Paul Beame*

1 Pseudorandomness and Sketching

Thus far we have been considered linear sketching algorithms for data streams where the sketch is given by an $S \times M$ random matrix A where S is small and at most the space bound and M is huge. So far, as in the Count-Min, Count, and Tug-of-War sketches, the matrix A has been defined implicitly using pairwise or 4-wise independence using a very small number of random bits that contribute only a small amount to the space bound. If we required truly M-wise independent random entries in A then it would cost too much to store them during the algorithm.

Today we will discuss a general method that will allow us to design algorithms assuming such independence and then show how to modify them to work using many fewer random bits, indeed few enough that we can store them all.

The key properties that we will use are

- Computing $A \cdot f$ requires storing very few bits that depend on the input. That is, except for the random bits, the algorithm uses very little space.
- The function $A \cdot f$ is independent of the order of the input stream.

Write the matrix $A = (a_{ij})_{ij}$. The i-th element of $A \cdot f$ is $\sum_{j=1}^{m} a_{ij} f_j$. Because the number of rows is small we focus on a single row of $A \cdot f$, dropping the index i. That is we focus on computing

$$a_1 \cdot f_1 + \cdots + a_M \cdot f_M$$

where $a_j = F_j(r_j) \in [-B, B]$ for some integer B, each r_j is a uniformly and independently chosen element of $\{0,1\}^K$ for some integer $K = O(\log B)$ and each F_j is some fixed, easy-to-compute function.

This row of A would take KM random bits to specify but we will see how to get nearly the same results using many fewer random bits.

PSeudorandom generators for small space

We will think about a finite state machine with a small number of states that tests potential random strings as inputs. A space S finite state machine has a state set Q of size 2^S. It has a state transition function given by a $2^S \times 2^S$ matrix T that depends on its input string r, which we think of as partitioned into blocks of $K = O(S)$ bits each that determine the entries of T. The finite state machine will also have a subset $P \subset Q$ of accepting states.

For states $p, q \in Q$ and $T_{p,q}(r)$ will determine whether the machine moves from state q to state p and will depend on precisely one block of r. We assume without loss of generality that the time step t is encoded in each state so that if the time step in state q is t, then the t-th block of r will determine the value of $T_{p,q}(r)$.

Let $Q^{(t)}(r)$ be the random variable denoting the state Q after t steps using randomness r starting in the initial state, $Q^{(0)}$. It is easy to check that $Q^{(t)}(r) = T(r)^t Q^{(0)}$.

We note that our usual space-bounded randomized algorithms have two inputs, a non-random input x and a randomly chosen string r, whereas the space-bounded does not have an x input. However, the above model also can encode the x input: For $t \in [n]$, the input x_t is hard-coded in the function that determines the transitions from states q with time $t - 1$ to time t. Therefore, each fixed input string corresponds to a separate finite state machine.

The above finite state machines are called *space S statistical tests*.

Definition 1.1. *A function $G : \{0,1\}^L \rightarrow \{0,1\}^R$ ε-*fools *statistical test A on R-bit strings iff*

$$|\mathbb{P}_{r \in_R \{0,1\}^R}[A(r) \textit{ accepts}] - |\mathbb{P}_{s \in_R \{0,1\}^L}[A(G(s)) \textit{ accepts}]| < \varepsilon.$$

s is called the seed *for G.*

G is called a *pseudorandom generator* if $L < R$ and G ϵ-fools a class of statistical tests for sufficiently small ε.

Nisan's Generator Nisan developed a pseudorandom generator for space-bounded machines whose properties are given by the following theorem.

Theorem 1.2 (Nisan 92)**.** *There is a constant $c > 0$ such that the following holds. For $L \leq 2S \log_2 R + S$ there is a mapping $G : \{0,1\}^L \rightarrow \{0,1\}^{SR}$ computable using $\log_2 R$ arithmetic operations on S-bit integers such that for every space S statistical test,*

$$|\mathbb{P}_{r \in_R \{0,1\}^{SR}}[Q^{(R)}(r) \textit{ is accepting}] - |\mathbb{P}_{s \in_R \{0,1\}^L}[Q^{(R)}(G(s)) \textit{ is accepting}]| < 2^{-cS}.$$

That is, the function G 2^{-cS}-fools all space S statistical tests runining in time R.

The construction of G takes $\log_2 R$ hash functions $h_1, \ldots, h_{\log_2 R}$ chosen independently from a family H of pairwise independent hash functions from $\{0,1\}^S$ to $\{0,1\}^S$ as well as a single input $x \in \{0,1\}^S$.

The output of the generator G is computed using a tree with x at its root. Each node at level i has two children: the left child is labeled by the same string as its parent, the right child is labeled by the string obtained by applying the hash function h_i to its parent string. The output string is the left-to-right concatenation of all the strings of the leaves at level $\log_2 R$, which is of length SR. For example, at level 3 the string would be

$$x\ h_3(x)\ h_2(x)\ h_3(h_2(x))\ h_1(x)\ h_3(h_1(x))\ h_2(h_1(x))\ h_3(h_2(h_1(x))).$$

Though we will not have time to go through the proof of this theorem, the basic idea of the argument is based on the repeated structure of pairs $x\ h(x)$ in the tree. We can imagine the set A of random strings that take a machine from one state p to q and another set B of random strings that take a machine from state q to state r.

Definition 1.3. *We say that* $h : \{0,1\}^S \to \{0,1\}^S$ *is* (ε, A, B)-good *iff*

$$\left| \mathbb{P}_{x \in_R \{0,1\}^S}[x \in A \textit{ and } h(x) \in B] - \frac{|A||B|}{2^{2s}} \right| \leq \varepsilon.$$

Thus h is (ε, A, B)-good if and only if the pair $(x, h(x))$ is within ε of behaving like a truly random string with respect to the set $A \times B$. The key property of pairwise independent hash functions that is used in the proof for the Nisan generator is the following:

Lemma 1.4 (Hash Mixing Lemma)**.** *Let* $h : \{0,1\}^S \to \{0,1\}^S$ *be a chosen from a pairwise independent hash function family* H *and* $A, B \subseteq \{0,1\}^S$*. Then*

$$\mathbb{P}_{h \sim H}[h \textit{ is not } (\varepsilon, A, B) - \textit{good}] \leq \varepsilon$$

for $\varepsilon \leq 2^{-S/3}$.

Applying the Nisan generator to sketching As we have defined things so far, the algorithm makes independent choices and uses each for only one time step. However, in the case of sketching we want to use a_j repeatedly whenever there is a j in the input, so these are clearly not independent and would need to be stored for later, which would be too large. However, there is an easy observation for this also. Because the output is independent of the order of the inputs, it would have the same output behavior on a sorted version of the input, $j_1, \ldots, j_1, j_2, \ldots, j_2, \ldots, j_\ell, \ldots, j_\ell$ where the inputs are in increasing order. In this case the algorithm only needs to store the current index of the input and it is clear that the is a segment that depends on r_{j_1}, followed by one depending on r_{j_2} etc. Therefore we apply the Nisan generator with $R = M$ and recompute the entry j as needed when $x_i = j$.

The original application of Nisan's generator to sketching was due to Indyk, who showed how to estimate F_p (or equivalently $||f||_p$) for $0 \leq p \leq 2$. The idea was to use a *p-stable* distribution: A random variable X is p-stable iff for any vector z and $X_j \sim X$, $\sum_{j=1}^{M} z_j X_j \sim ||z||_p X$. For example, the Gaussian distribution is a p-stable distribution for $p = 2$. We will instead consider a slightly different problem.

A constant-factor approximation for F_p (or $||f||_p$) for $p > 2$ The original Alon, Matias, and Szegedy paper was the first to give an approximation algorithm for estimating F_p using space $M^{1-1/p} \log^{O(1)} M$. They also showed a space lower bound of $M^{1-4/p}$. This was improved by Indyk and Woodruff in 2005 [] to $M^{1-2/p} \log^{O(1)} M$ which is tight up to log factors. This was later sharpened by Andoni, Krauthgammer, and Onak in 2010 [] to yield a simpler algorithm with a better log factor. The algorithm we describe is in a manuscript due to Andoni in 2012 and uses the above ideas together with some from a paper by Jowhari, Saglam, and Tardos for sampling elements approximately proportional to their contribution to the ℓ_p mass of their frequencies for $p \in [1, 2]$.

Though no p-stable distributions exist for $p > 2$, the general idea of the argument is to use a distribution that has a property similar to stability with respect to taking the maximum.

The *exponential distribution* $Exp(1)$ is given by

$$\mathrm{P}[x > t] = e^{-t} \qquad \text{for } t \geq 0.$$

One can check that it is indeed a distribution by noting that $\int_0^\infty e^{-t} \mathrm{d}t = -e^{-t}|_0^\infty = -(0-1) = 1$.

The General Idea The general idea of the algorithm will be to rescale each f_j using the exponential distribution to get z_j so that with good probability $||z||_\infty$ is a constant factor approximation to $||f||_p$. The algorithm will run a variant of the Count sketch on z in order to approximate $||z||_\infty$. Though the Count sketch does not yield constant factor approximations in general, z will have sufficient structure that we will be able to argue that this variant does work well. Though the space won't be too large, the entire algorithm as described so far would use a huge number of random bits, but we will apply the Nisan generator to the result.

More precisely, for each j, let $z_j = f_j / u_j^{1/p}$ where $u_j \sim Exp(1)$ are chosen independently.

Claim 1: $\mathbb{P}[\frac{||f||_p}{2} \leq ||z||_\infty \leq 2||f||_p] > 3/4$.

Proof. Let $q = 1/|z||_\infty^p$. Then

$$q = \frac{1}{\max_j \{|f_j|^p / u_j\}} = \min\{\frac{u_1}{|f_j|^p}, \dots \frac{u_M}{|f_M|^p}\}.$$

Let $\lambda \geq 0$. Then

$$\begin{aligned}
\mathbb{P}[q > \lambda] &= \mathbb{P}[\forall j \in [M],\ \frac{u_j}{|f_j|^p} > \lambda] \\
&= \prod_{j=1}^{M} \mathbb{P}[\frac{u_j}{|f_j|^p} > \lambda] \qquad \text{since the } u_j \text{ are independent} \\
&= \prod_{j=1}^{M} e^{-\lambda |f_j|^p} \\
&= e^{-\lambda \sum_{j=1}^{M} |f_j|^p} \\
&= e^{-\lambda ||f_j||_p^p}
\end{aligned}$$

Therefore $u = q||f||_p^p \sim Exp(1)$. It follows that

$$\begin{aligned}
&\mathbb{P}[\frac{||f||_p}{2} \leq ||z||_\infty \leq 2||f||_p] \\
&\quad = \mathbb{P}[\frac{||f||_p^p}{2^p} \leq ||z||_\infty^p \leq 2^p ||f||_p^p] \\
&\quad = \mathbb{P}[\frac{||f||_p^p}{2^p} \leq \frac{1}{q} \leq 2^p ||f||_p^p] \\
&\quad = \mathbb{P}[\frac{1}{2^p ||f||_p^p} \leq q \leq \frac{2^p}{||f||_p^p}] \\
&\quad = \mathbb{P}[\frac{1}{2^p} \leq q||f||_p^p \leq 2^p] \\
&\quad = \mathbb{P}[\frac{1}{2^p} \leq u \leq 2^p] \\
&\quad = e^{-1/2^p} - e^{-2^p} \\
&\quad > 3/4
\end{aligned}$$

since $e^{-1/2^p} - e^{-2^p} \geq e^{-1/4-4} = 0.76...$ for $p > 2$. $\square$

In the next class we will complete the description of the algorithm and its proof.