CS 598CSC: Algorithms for Big Data	**Lecture date: August 26, 2014**
Instructor: Chandra Chekuri	**Scribe: Chandra Chekuri**

1 Introduction/Administrivia

- Course website: `https://courses.engr.illinois.edu/cs598csc/fa2014/`.
- There is no specific book that we will follow. See website for pointers to several resources.
- Grading based on 4-5 homeworks, scribing a lecture and course project. Details to be figured out.

Course Objectives

Big Data is all the rage today. The goal of this course is learn about some of the basic algorithmic and analysis techniques that have been useful in this area. Some of the techniques are old and many have been developed over the last fifteen years or so. A few topics that we hope to cover are:

- Streaming, Sketching and Sampling
- Dimensionality Reduction
- Streaming for Graphs
- Numerical Linear Algebra
- Compressed Sensing
- Map-Reduce model and some basic algorithms
- Property Testing
- Lower Bounds via Communication Complexity

There is too much material in the above topics. The plan is to touch upon the basics so that it will provide an impetus to explore further.

2 Streaming/One-Pass Model of Computation

In the streaming model we assume that the input data comes as a stream. More formally, the data is assumed to consist of m items/tokens/objects $a_1, a_2, \ldots, a_m$. A simple case is when each token is a number in the range $[1..n]$. Another example is when each token represents an edge in a graph given as (u, v) where u and and v are integers representing the node indices. The tokes arrive one by one in order. The algorithm has to process token x_i before it sees the next token. If we are allowed to store all the tokens then we are in the standard model of computing where we have access to the whole input. However, we will assume that the space available to the algorithm is much less than m tokens; typically sub-linear in m or in the ideal scenario polylog(m). Our goal is to (approximately) compute/estimate various quantities of interest using such limited space. Surprisingly one can compute various useful functions of interest. Some parameters of interest for a streaming algorithm are:

- space used by the algorithm as a function of the stream length m and the nature of the tokens
- the worst-case time to process each token
- the total time to process all the tokens (equivalently the amortized time to process a token)
- the accuracy of the output
- the probability of success in terms of obtaining a desired approximation (assuming that the algorithm is randomized)

There are several application areas that motivate the streaming model. In networking, a large number of packets transit a switch and we may want to analyze some statistics about the packets. The number of packets transiting the switch is vastly more than what can be stored for offline processing. In large databases the data is stored in disks and random access is not feasible; the size of the main memory is much smaller than the storage available on the disk. A one-pass or multi-pass streaming algorithm allows one to avoid sophisticated data structures on the disk. There are also applications where the data is distributed in several places and we need to process them separately and combine the results with low communication overhead.

In database applications and such it also makes sense to discuss the multi-pass model or the semi-streaming model where one gets to make several passes over the data. Typically we will assume that the number of passes is a small constant.

3 Background on Probability and Inequalities

The course will rely heavily on proababilistic methods. We will mostly rely on discrete probability spaces. We will keep the discussion high-level where possible and use certain results in a black-box fashion.

Let Ω be a finite set. A probability measure p assings a non-negative number $p(\omega)$ for each $\omega \in \Omega$ such that $\sum_{\omega \in \Omega} p(\omega) = 1$. The tuple (Ω, p) defines a discrete probability space; an event in this space is any subset $A \subseteq \Omega$ and the probability of an event is simply $p(A) = \sum_{\omega \in A} p(\omega)$. When Ω is a continuous space such as the interval $[0, 1]$ things get trickier and we need to talk about a measure spaces σ-algebras over Ω; we can only assign probability to certain subsets of Ω. We will not go into details since we will not need any formal machinery for what we do in this course.

An important definition is that of a *random variable.* We will focus only on real-valued random variables in this course. A random variable X in a probability space is a function $X : \Omega \to \mathbb{R}$. In the discrete setting the *expectation* of X, denoted by $\mathbf{E}[X]$, is defined as $\sum_{\omega \in \Omega} p(w)X(\omega)$. For continuous spaces $\mathbf{E}[X] = \int X(\omega)dp(\omega)$ with appropriate definition of the integral. The variance of X, denoted by $\mathbf{Var}[X]$ or as σ_X^2, is defined as $\mathbf{E}[(X - \mathbf{E}[X])^2]$. The standard deviation is σ_X, the square root of the variance.

Theorem 1 (Markov's Inequality) *Let X be a non-negative random variable such that $\mathbf{E}[X]$ is finite. Then for any $t > 0$, $\Pr[X \geq t] \leq \mathbf{E}[X]/t$.*

Proof: The proof is in some sense obvious, especially in the discrete case. Here is a sketch. Define a new random variable Y where $Y(\omega) = X(\omega)$ if $X(\omega) < t$ and $Y(\omega) = t$ if $X(\omega) \geq t$. Y is non-negative and $Y \leq X$ point-wise and hence $\mathbf{E}[Y] \leq \mathbf{E}[X]$. We also see that:

$$\begin{aligned}\mathbf{E}[X] \geq \mathbf{E}[Y] &= \sum_{\omega: X(\omega) < t} X(\omega)p(\omega) + \sum_{\omega: X(\omega) \geq t} tp(\omega) \\ &\geq t \sum_{\omega: X(\omega) \geq t} p(\omega) \qquad \text{(since } X \text{ is non-negative)} \\ &\geq t \Pr[X \geq t].\end{aligned}$$

The continuous case follows by replacing sums by integrals. □

Markov's inequality is tight under the assumption. Assume you can construct an example. The more information we have about a random variable the better we can bound its deviation from the expectation.

Theorem 2 (Chebyshev's Inequality) *Let X be a random variable with $\mathbf{E}[X]$ and $\mathbf{Var}[X]$ finite. Then $\Pr[|X| \geq t] \leq \mathbf{E}[X^2]/t^2$ and $\Pr[|X - \mathbf{E}[X]| \geq t\sigma_X] \leq 1/t^2$.*

Proof: Consider the non-negative random variable $Y = X^2$. $\Pr[|X| \geq t] = \Pr[Y \geq t^2]$ and we apply Markov's inequality to the latter. The second inequality is similar by considering $Y = (X - \mathbf{E}[X])^2$. □

Chernoff-Hoeffding Bounds: We will use several times various forms of the Chernoff-Hoeffding bounds that apply to a random variable that is a a finite sum of bounded and *independent* random variables. There are several versions of these bounds. First we state a general bound that is applicable to non-negative random variables and is dimension-free in that it depends only the expectation rather than the number of variables.

Theorem 3 (Chernoff-Hoeffding) *Let $X_1, X_2, \ldots, X_n$ be independent binary random variables and let $a_1, a_2, \ldots, a_n$ be coefficients in $[0, 1]$. Let $X = \sum_i a_i X_i$. Then*

- *For any $\mu \geq \mathbf{E}[X]$ and any $\delta > 0$, $\Pr[X > (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$.*
- *For any $\mu \leq \mathbf{E}[X]$ and any $\delta > 0$, $\Pr[X < (1 - \delta)\mu] \leq e^{-\mu\delta^2/2}$.*

The following corollary bounds the deviation from the mean in both directions.

Corollary 4 *Under the conditions of Theorem 3, the following hold:*

- *If $\delta > 2e - 1$, $\Pr[X \geq (1 + \delta)\mu] \leq 2^{-(1+\delta)\mu}$.*
- *For any U there is a constant $c(U)$ such that for $0 < \delta < U$, $\Pr[X \geq (1 + \delta)\mu] \leq e^{-c(U)\delta^2\mu}$. In particular, combining with the lower tail bound,*

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-c(U)t^2\mu}.$$

We refer the reader to the standard books on randomized algorithms [6] and [4] for the derivation of the above bounds.

If we are interested only in the upper tail we also have the following bounds which show the dependence of μ on n to obtain an inverse polynomial probability.

Corollary 5 *Under the conditions of Theorem 3, there is a universal constant α such that for any $\mu \geq \max\{1, \mathbf{E}[X]\}$, and sufficiently large n and for $c \geq 1$, $\Pr[X > \frac{\alpha c \ln n}{\ln \ln n} \cdot \mu] \leq 1/n^c$. Similarly, there is a constant α such that for any $\epsilon > 0$, $\Pr[X \geq (1+\epsilon)\mu + \alpha c \log n/\epsilon] \leq 1/n^c$.*

Remark 6 *If the X_i are in the range $[0, b]$ for some b not equal to 1 one can scale them appropriately and then use the standard bounds.*

Some times we need to deal with random variables that are in the range $[-1, 1]$. Consider the setting where $X = \sum_i X_i$ where for each i, $X_i \in [-1, 1]$ and $\mathbf{E}[X_i] = 0$, and the X_i are independent. In this case $\mathbf{E}[X] = 0$ and we can no longer expect a dimension-free bound. Suppose each X_i is 1 with probability 1/2 and -1 with probability 1/2. Then $X = \sum_i X_i$ corresponds to a 1-dimensional random walk and even though the expected value is 0 the standard deviation of X is $\Theta(\sqrt{n})$. One can show that $\Pr[|X| \geq t\sqrt{n}] \leq 2e^{-t^2/2}$. For these settings we can use the following bounds.

Theorem 7 *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that for each i, $X_i \in [a_i, b_i]$. Let $X = \sum_i a_i X_i$ and let $\mu = \mathbf{E}[X]$. Then*

$$\Pr[|X - \mu| \geq t] \leq 2e^{-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}}.$$

In particular if $b_i - a_i \leq 1$ for all i then

$$\Pr[|X - \mu| \geq t] \leq 2e^{-\frac{2t^2}{n}}.$$

Note that $\mathbf{Var}[X] = \sum_i \mathbf{Var}[X_i]$. One can show a bound based of the following form

$$\Pr[|X - \mu| \geq t] \leq 2e^{-\frac{t^2}{2(\sigma_X^2 + Mt/3)}}$$

where $|X_i| \leq M$ for all i.

Remark 8 *Compare the Chebyshev bound to the Chernoff-Hoeffding bounds for the same variance.*

Statistical Estimators, Reducing Variance and Boosting: In streaming we will mainly work with randomized algorithms that compute a function f of the data stream $x_1, \ldots, x_m$. They typically work by producing an unbiased estimator, via a random variable X, for the the function value. That is, the algorithm will have the property that the $\mathbf{E}[X]$ is the desired value. Note that the randomness is internal to the algorithm and not part of the input (we will also later discuss randomness in the input when considering random order streams). Having an estimator is not often useful. We will also typically try to evaluate $\mathbf{Var}[X]$ and then we can use Chebyshev's inequality. One way to reduce the variance of the estimate is to run the algorithm in parallel (with separate random bits) and get estimators $X_1, X_2, \ldots, X_h$ and use $X = \frac{1}{h}\sum_i X_i$ as the final estimator. Note that $\mathbf{Var}(X) = \frac{1}{h}\sum_i \mathbf{Var}(X_i)$ since the X_i are independent. Thus the variance has been reduced by a factor of h. A different approach is to use the *median* value of $X_1, X_2, \ldots, X_h$ as the final estimator. We can then use Chernoff-Hoeffding bounds to get a much better dependence on h. In fact both approaches can be combined and we illustrate it via a concrete example in the next section.

4 Probabilistic Counting and Morris's Algorithm

Suppose we have a long sequence of events that we wish to count. If we wish to count a sequent of events of length upto some N we can easily do this by using a counter with $\lceil \log_2 N \rceil$ bits. In some settings of interest we would like to further reduce this. It is not hard to argue that if one wants an exact and deterministic count then we do need a counter with $\lceil \log_2 N \rceil$ bits. Surprisingly, if we allow for approximation and randomization, one can count with about $\log logN$ bits. This was first shown in a short and sweet paper of Morris [5]; it is a nice paper to read the historical motivation.

Morris's algorithm keeps a counter that basically keeps an estimate of $\log N$ where N is the number of events and this requires about $\log logN$ bits. There are several variants of this, here we will discuss the simple one and point the reader to [5, 3, 1] for other schemes and a more detailed and careful analysis.

```
ProbabilisticCounting:
$X \leftarrow 0$
While (a new event arrives)
  Toss a biased coin that is heads with probability $1/2^X$
  If (coin turns up heads)
      $X \leftarrow X + 1$
endWhile
Output $2^X - 1$ as the estimate for the length of the stream.
```

For integer $i \geq 0$, let X_n be the random variable that denotes the value of the counter after i events. Let $Y_n = 2^{X_n}$. The lemma below shows that the output of the algorithm is an unbiased estimator of the count that we desire.

Lemma 9 $\mathbf{E}[Y_n] = n + 1$.

Proof: Proof by induction on n. The case of $n = 0, 1$ is easy to verify since in both cases we have that Y_n is deterministically equal to $n + 1$. We have for $n \geq 2$:

$$\begin{aligned}
\mathbf{E}[Y_n] = \mathbf{E}[2^{X_n}] &= \sum_{j=0}^{\infty} 2^j \Pr[X_n = j] \\
&= \sum_{j=0}^{\infty} 2^j \left(\Pr[X_{n-1} = j] \cdot (1 - \frac{1}{2^j}) + \Pr[X_{n-1} = j-1] \cdot \frac{1}{2^{j-1}} \right) \\
&= \sum_{j=0}^{\infty} 2^j \Pr[X_{n-1} = j] + \sum_{j=0}^{\infty} (2 \Pr[X_{n-1} = j-1] - \Pr[X_{n-1} = j]) \\
&= \mathbf{E}[Y_{n-1}] + 1 \\
&= n + 1
\end{aligned}$$

where we used induction to obtain the final equality. □

Since $\mathbf{E}[Y_n] = n + 1$ we have $\mathbf{E}[X_n] = \log_2(n + 1)$ which implies that the expected number of bits in the counter after n events is $\log \log n + O(1)$ bits. We should also calculate the variance of Y_n.

Lemma 10 $\mathbf{E}[Y_n^2] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$ *and* $\mathbf{Var}[Y_n] = n(n-1)/2$.

Proof: Proof is by induction on n. Easy to verify base cases $n = 0, 1$ since $Y_n = n + 1$ deterministically. For $n \geq 2$:

$$\begin{aligned}
\mathbf{E}[Y_n^2] = \mathbf{E}[2^{2X_n}] &= \sum_{j\geq 0} 2^{2j} \cdot \Pr[X_n = j] \\
&= \sum_{j\geq 0} 2^{2j} \cdot \left(\Pr[X_{n-1} = j](1 - \frac{1}{2^j}) + \Pr[X_{n-1} = j-1]\frac{1}{2^{j-1}}\right) \\
&= \sum_{j\geq 0} 2^{2j} \cdot \Pr[X_{n-1} = j] + \sum_{j\geq 0} \left(-2^j \Pr[X_{n-1} = j-1] + 42^{j-1} \Pr[X_{n-1} = j-1]\right) \\
&= \mathbf{E}[Y_{n-1}^2] + 3\mathbf{E}[Y_{n-1}] \\
&= \frac{3}{2}(n-1)^2 + \frac{3}{2}(n-1) + 1 + 3n \\
&= \frac{3}{2}n^2 + \frac{3}{2}n + 1.
\end{aligned}$$

We used induction and the value of $\mathbf{E}[Y_n]$ that we previously computed. $\mathbf{Var}[Y_n] = \mathbf{E}[Y_n^2] - (\mathbf{E}[Y_n])^2$ and it can be verified that it is equal to $n(n-1)/2$. □

4.1 Approximation and Success Probability

The analysis of the expectation shows that the output of the algorithm is an unbiased estimator. The analysis of the variance shows that the estimator is fairly reasonable. However, we would like to have a finer understanding. For instance, given some parameter $\epsilon > 0$, what is $\Pr[|Y_n - (n+1)| > \epsilon n]$? We could also ask a related question. Is there an algorithm that given ϵ, δ guarantees that the output Y will satisfy the property that $\Pr[|Y - n| > \epsilon n] \leq \delta$ while still ensuring that the counter size is $O(\log \log n)$; of course we would expect that the constant in the $O()$ notation will depend on ϵ, δ.

The algorithm can be modified to obtain a $(1 + \epsilon)$-approximation with constant probability using a related scheme where the probability of incrementing the counter is $\frac{1}{a^X}$ for some parameter a; see [5, **?**]. The expected number of bits in the counter to achieve a $(1 + \epsilon)$-approximation can be shown to be $\log \log n + O(\log 1/\epsilon)$ bits.

Here we describe two general purpose ways to obtain better approximations by using independent estimators. Suppose we run the basic algorithm h times in parallel with independent randomness to get estimators $Y_{n,1}, Y_{n,2}, \ldots, Y_{n,h}$. We then output $Z = \frac{1}{h}\sum_{i=1}^{h} Y_{n,i} - 1$ as our estimate for n. Note that Z is also an unbiased estimator. We have that $\mathbf{Var}[Z] = \frac{1}{h}\mathbf{Var}[Y_n]$.

Claim 11 *Suppose* $h = 2/\epsilon^2$ *then* $\Pr[|Z - n| \geq \epsilon n] < \frac{1}{4}$.

Proof: We apply Chebyshev's inequality to obtain that

$$\Pr[|Z - n| \geq \epsilon n] \leq \frac{\mathbf{Var}[Z]}{\epsilon^2 n^2} \leq \frac{1}{h}\frac{\mathbf{Var}[Y_n]}{\epsilon^2 n^2} \leq \frac{1}{h}\frac{n(n-1)}{2\epsilon^2 n^2} < \frac{1}{4}.$$

□

Now suppose we want a high probability guarantee regarding the approximation. That is, we would like the estimator to be a $(1 + \epsilon)$-approximation with probability at least $(1 - \delta)$ for some given parameter δ.

We choose $\ell = c \log \frac{1}{\delta}$ for some sufficiently large constant c. We independently and in parallel obtain estimators $Z_1, Z_2, \ldots, Z_\ell$ and output the *median* of the estimators; lets call Z' the random variable corresponding to the median. We will prove the following.

Claim 12 $\Pr[|Z' - n| \geq \epsilon n] \leq (1 - \delta)$.

Proof: Define an indicator random variable A_i, $1 \leq i \leq \ell$ where A_i is 1 if $|Z_i - n| \geq \epsilon n$. From Claim 11, $\Pr[A_i = 1] < 1/4$. Let $A = \sum_{i=1}^{\ell} A_i$ and hence $\mathbf{E}[A] < \ell/4$. We also observe that $|Z' - n| \geq \epsilon n$ only if $A \geq \ell/2$, that is, if more than half of the Z_i's deviate by more than ϵn. We can apply Chernoff-Hoeffding bound to upper bound $\Pr[A \geq \ell/2] = \Pr[A \geq (1+\delta)\mu]$ where $\delta = 1$ and $\mu = \ell/4 \geq \mathbf{E}[A]$. From Theorem 3 this is at most $(e/4)^{\ell/4}$. Since $\ell = c \log \frac{1}{\delta}$, the probability is at most δ for an appropriate choice of c. □

The total space usage for running the estimates in parallel is $O(\frac{1}{\epsilon^2} \cdot \log \frac{1}{\delta} \cdot \log \log n)$.

5 Reservoir Sampling

Random sampling is a powerful general purpose technique in a variety of areas. The goal is to pick a small subset S of a set of items N such that S is *representative* of N and often a random sample works well. The simplest sampling procedure is to pick a *uniform* sample of size k from a set of size m where $k \leq m$ (typically $k \ll m$). We can consider sampling with or without replacement. These are standard and easy procedures if the whole data set is available in a random-access manner — here we are assuming that we have access to a random number generator.

Below we describe a simple yet nice technique called reservoir sampling to obtain a uniform sample of size 1 from a stream.

```
UniformSample:
$s \leftarrow$ null
$m \leftarrow 0$
While (stream is not done)
  $m \leftarrow m + 1$
  $x_m$ is current item
  Toss a biased coin that is heads with probability $1/m$
  If (coin turns up heads)
      $s \leftarrow x_m$
endWhile
Output $s$ as the sample
```

Lemma 13 *Let m be the length of the stream. The output of the algorithm s is uniform. That is, for any $1 \leq j \leq m$, $\Pr[s = x_j] = 1/m$.*

Proof: We observe that $s = x_j$ if x_j is chosen when it is considered by the algorithm (which happens with probability $1/j$), and none of $x_{j+1}, \ldots, x_m$ are chosen to replace x_j. All the relevant events are independent and we can compute:

$$\Pr[s = x_j] = 1/j \times \prod_{i>j} (1 - 1/i) = 1/m.$$

□

To obtain k samples *with* replacement, the procedure for $k = 1$ can be done in parallel with independent randomness. Now we consider obtaining k samples from the stream *without* replacement. The output will be stored in an array of S of size k.

```
SAMPLE-WITHOUT-REPLACEMENT($k$):

$S[1..k] \leftarrow$ null
$m \leftarrow 0$
While (stream is not done)
  $m \leftarrow m + 1$
  $x_m$ is current item
  If ($m \leq k$)
      $S[m] \leftarrow x_m$
  Else
      $r \leftarrow$ uniform random number in range $[1..m]$
      If ($r \leq k$)
          $S[r] \leftarrow x_m$
endWhile
Output $S$
```

An alternative description is that when item x_t arrives (for $t > k$) we decide to choose it for inclusion in S with probability k/t, and if it is chosen then we choose a uniform element from S to be replaced by x_t.

Exercise: Prove that the algorithm outputs a random sample without replacement of size k from the stream.

Weighted sampling: Now we consider a generalization of the problem to weighted sampling. Suppose the stream consists of m items $x_1, \ldots, x_m$ and each item j has a weigth $w_j > 0$. The goal is to choose a k sample in proportion to the weights. Suppose $k = 1$ then the goal is to choose an item s such that $\Pr[s = x_j] = w_j/W$ where $W = \sum_i w_i$. It is easy to generalize the uniform sampling algorithm to achieve this and k samples with replacement is also easy. The more interesting case is when we want k samples without replacement. The precise definition of what this means is not so obvious. Here is an algorithm. Obtain first a uniform sample s from $x_1, \ldots, x_m$ in proportion to the weights. Remove s from the set and obtain another uniform sample in the residual set. Repeat k times to obtain a set of k items (assume that $k < m$). The k removed items form the output S. We now want to obtain a random set S according to this same distribution but in a streaming fashion.

First we describe a randomized offline algorithm below.

```
WEIGHTED-SAMPLE-WITHOUT-REPLACEMENT($k$):

For $i = 1$ to $m$ do
  $r_i \leftarrow$ uniform random number in interval $(0, 1)$
  $w'_i = r_i^{1/w_i}$
endFor
Sort items in decreasing order according to $w'_i$ values
Output the first $k$ items from the sorted order
```

We leave it as a simple exercise to show that the above algorithm can be implemented in the stream model by keeping the heaviest k modified weights seen so far. Now for the analysis.

To get some intuition we make the following claim.

Claim 14 *Let r_1, r_2 be independent unformly distributed random variables over $[0,1]$ and let $X_1 = r_1^{1/w_1}$ and $X_2 = r_2^{1/w_2}$ where $w_1, w_2 \geq 0$. Then*

$$\Pr[X_1 \leq X_2] = \frac{w_2}{w_1 + w_2}.$$

The above claim can be shown by doing basic analysis via the probability density functions. More precisely, suppose $w > 0$. Consider the random variable $Y = r^{1/w}$ where r is chosen uniformly in $(0,1)$. The cumulative probabilty function of Y,

$$F_Y(t) = \Pr[Y \leq t] = \Pr[r^{1/w} \leq t] = \Pr[r \leq t^w] = t^w.$$

Therefore the density function $f_Y(t)$ is wt^{w-1}. Thus

$$\Pr[X_1 \leq X_2] = \int_0^1 F_{Y_1}(t) f_{Y_2}(t) dt = \int_0^1 t^{w_1} w_2 t^{w_2 - 1} dt = \frac{w_2}{w_1 + w_2}.$$

We now make a much more general statement.

Lemma 15 *Let $r_1, r_2, \ldots, r_n$ be independent random variables each of which is uniformly distributed random variables over $[0,1]$. Let $X_i = r_i^{1/w_i}$ for $1 \leq i \leq n$. Then for any $\alpha \in [0,1]$*

$$\Pr[X_1 \leq X_2 \ldots \leq X_n \leq \alpha] = \alpha^{w_1 + w_2 + \ldots + w_n} \cdot \prod_{i=1}^{n} \frac{w_i}{w_1 + \ldots + w_i}.$$

Proof: By induction on n. For $n = 1$, $\Pr[X_1 \leq \alpha] = F_{Y_1}(\alpha) = \alpha^{w_1}$. Assuming the lemma holds for all $h < n$ we prove it for n.

$$\begin{aligned}
\Pr[X_1 \leq \ldots \leq X_n \leq \alpha] &= \int_0^\alpha \Pr[X_1 \leq \ldots \leq X_{n-1} \leq t] f_{Y_n}(t) dt \\
&= \int_0^\alpha t^{w_1 + w_2 + \ldots + w_{n-1}} \cdot \left(\prod_{i=1}^{n-1} \frac{w_i}{w_1 + \ldots + w_i} \right) w_n t^{w_n - 1} dt \\
&= w_n \left(\prod_{i=1}^{n-1} \frac{w_i}{w_1 + \ldots + w_i} \right) \int_0^\alpha t^{w_1 + w_2 + \ldots + w_n - 1} dt \\
&= \alpha^{w_1 + w_2 + \ldots + w_n} \cdot \prod_{i=1}^{n} \frac{w_i}{w_1 + \ldots + w_i}.
\end{aligned}$$

We used the induction hypothesis in the second equality. □

Now we are ready to finish the proof. Consider any fixed j. We claim that the probability that X_j is the largest number among $X_1, X_2, \ldots, X_m$ is equal to $\frac{w_j}{w_1 + \ldots + w_n}$. Do you see why? Conditioned on X_j being the largest, the rest of the variables are still independent and we can apply this observation again. You should hopefully be convinced that picking the largest k among the values $X_1, X_2, \ldots, X_m$ gives the desired sample.

Bibliographic Notes: Morris's algorithm is from [5]. See [3] for a detailed analysis and [1] for a more recent treatment which seems cleaner and easier. Weighted reservoir sampling is from [2]. See [7] for more on efficient reservoir sampling methods.

References

[1] Miklós Csürös. Approximate counting with a floating-point counter. *CoRR*, abs/0904.3062, 2009.

[2] Pavlos S Efraimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[3] Philippe Flajolet. Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985.

[4] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

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

[6] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[7] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

CS 598CSC: Algorithms for Big Data	Lecture date: August 28, 2014
Instructor: Chandra Chekuri	Scribe: Chandra Chekuri

1 Estimating Frequency Moments in Streams

A significant fraction of streaming literature is on the problem of estimating frequency moments. Let $\sigma = a_1, a_2, \ldots, a_m$ be a stream of numbers where for each i, a_i is an intger between 1 and n. We will try to stick to the notation of using m for the length of the stream and n for range of the integers[1]. Let f_i be the number of occurences (or frequency) of integer i in the stream. We let $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ be the frequency vector for a given stream σ. For $k \geq 0$, $F_k(\sigma)$ is defined to be the k'th frequency moment of σ:

$$F_k = \sum_i f_i^k.$$

We will discuss several algorithms to estimate F_k for various values of k. For instance F_0 is simply the number of distinct elements in σ. Note that $F_1 = \sum_i f_i = m$, the length of the stream. A k increases to ∞ F_k will concentrate on the most frequent element and we can thing of F_∞ as finding the most frequent element.

Definition 1 *Let $\mathcal{A})(\sigma)$ be the real-valued output of a randomized streaming algorithm on stream σ. We say that $\mathcal{A}$ provides an (α, β)-approximation for a real-valued function g if*

$$\Pr\left[|\frac{\mathcal{A}(\sigma)}{g(\sigma)} - 1| > \alpha\right] \leq \beta$$

for all σ.

Our ideal goal is to obtain a (ϵ, δ)-approximation for any given $\epsilon, \delta \in (0, 1)$.

2 Background on Hashing

Hashing techniques play a fundamental role in streaming, in particular for estimating frequency moments. We will briefly review hashing from a theoretical point of view and in particular k-universal hashing.

A hash function maps a finite universe $\mathcal{U}$ to some range $\mathcal{R}$. Typically the range is the set of integers $[0..L-1]$ for some finite L (here L is the number of buckets in the hash table). Sometimes it is convenient to consider, for the sake of developing intuition, hash functions that maps $\mathcal{U}$ to the continuous interval $[0, 1]$. We will, in general, be working with a family of hash functions $\mathcal{H}$ and h will be drawn from $\mathcal{H}$ uniformly at random; the analyis of the algorithm will be based on properties of $\mathcal{H}$. We would like $\mathcal{H}$ to have two important and contradictory properties:

- a random function from $\mathcal{H}$ should behave like a completely random function from $\mathcal{U}$ to the range.
- $\mathcal{H}$ should have nice computational properties:

[1]Many streaming papers flip the notation and use n to denote the length of the stream and m to denote the range of the integers in the stream

– a uniformly random function from $\mathcal{H}$ should be easy to sample

– any function $h \in \mathcal{H}$ should have small representation size so that it can be stored compactly

– it should be efficient to evaluate h

Definition 2 *A collection of random variables $X_1, X_2, \ldots, X_n$ are k-wise independent if the variables $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$ are independent for any set of distinct indices $i_1, i_2, \ldots, i_k$.*

Take three random variables $\{X_1, X_2, X_3\}$ where X_1, X_2 are independent $\{0, 1\}$ random variables and $X_3 = X_1 \oplus X_2$. It is easy to check that the three variables are pairwise independent although they are not all independent.

Following the work of Carter and Wegman [3], the class of k-universal hash families, and in particular for $k = 2$, provide an excellent tradeoff. $\mathcal{H}$ is strongly 2-universal if the following properties hold for a random function h picked from $\mathcal{H}$: (i) for every $x \in \mathcal{U}$, $h(x)$ (which is a random variable) is uniformly distributed over the range and (ii) for every distinct pair $x, y \in \mathcal{U}$, $h(x)$ and $h(y)$ are independent. 2-universal hash families are also called pairwise independent hash families. A weakly 2-universal family satisfies the property that that $\Pr[h(x) = h(y)] = 1/L$ for any distinct x, y. We state an important observation about pairwise independent random variables.

Lemma 1 *Let $Y = \sum_{i=1}^{h} X_i$ where $X_1, X_2, \ldots, X_h$ are pairwise independent. Then*

$$\mathbf{Var}[Y] = \sum_{i=1}^{h} \mathbf{Var}[X_i].$$

Moreover if X_i are binary/indicator random variables then

$$\mathbf{Var}[Y] \leq \sum_i \mathbf{E}[X_i^2] = \sum_i \mathbf{E}[X_i] = \mathbf{E}[Y].$$

There is a simple and nice construction of pairwise independent hash functions. Let p be a prime number such that $p \geq |\mathcal{U}|$. Recall that $Z_p = \{0, 1, \ldots, p-1\}$ forms a field under the standard addition and multiplication modulo p. For each $a, b \in [p]$ we can define a hash function $h_{a,b}$ where $h_{a,b}(x) = ax + b \mod p$. Let $\mathcal{H} = \{h_{a,b} \mid a, b \in [p]\}$. We can see that we only need to store two numbers a, b of $\Theta(\log p)$ bits to implicitly store $h_{a,b}$ and evaluation of $h_{a,b}(x)$ takes one addition and one multiplication of $\log p$ bit numbers. Moreover, samply a random hash function from $\mathcal{H}$ requires sampling a, b which is also easy. We claim that $\mathcal{H}$ is a pairwise independent family. You can verify this by the observation that for distinct x, y and any i, j the two equations $ax + b = i$ and $ay + b = j$ have a unique a, b them simultaneously. Note that if $a = 0$ the hash function is pretty useless; all elements get mapped to b. Nevertheless, for $\mathcal{H}$ to be pairwise independent one needs to include those hash functions but the probability that $a = 0$ is $1/p$ while there are p^2 functions in $\mathcal{H}$. If one only wants a weakly universal hash family we can pick a from $[1..(p-1)]$. The range of the hash function is $[p]$. To restrict the range to L we let $h'_{a,b}(x) = (ax + b \mod p) \mod L$.

More generally we will say that $\mathcal{H}$ is k-universal if every element is uniformly distributed in the range and for any k elements $x_1, \ldots, x_k$ the random variabels $h(x_1), \ldots, h(x_k)$ are independent. Assuming $\mathcal{U}$ is the set of integers $[0..|\mathcal{U}|]$, for any fixed k there exist constructions for k-universal hash families such that every hash function h in the family can be stored using $O(k \log |\mathcal{U}|)$ bits (essentially k numbers) and h can be evaluated using $O(k)$ arithmetic operations on $\log |\mathcal{U}|$ bit numbers. We will ignore specific details of the implementations and refer the reader to the considerable literature on hashing for further details.

3 Estimating Number of Distinct Elements

A lower bound on exact counting deterministic algorithms: We argue that any deterministic streaming algorithm that counts the number of distinct elements exactly needs $\Omega(n)$ bits. To see this, suppose there is an algorithm $\mathcal{A}$ that uses strictly less than n bits. Consider the $h = 2^n$ different streams σ_S where $S \subseteq [n]$; σ_S consists of the elements of S in some arbitrary order. Since $\mathcal{A}$ uses $n-1$ bits or less, there must be two distinct sets S_1, S_2 such that the state of $\mathcal{A}$ at the end of $\sigma_{S_1}, \sigma_{S_2}$ is identical. Since S_1, S_2 are distinct there is an element i in $S_1 \setminus S_2$ or $S_2 \setminus S_1$; wlog it is the former. Then it is easy to see the $\mathcal{A}$ cannot give the right count for at least one of the two streams, $< \sigma_{S_1}, i >$, $< \sigma_{S_2}, i >$.

The basic hashing idea: We now discuss a simple high-level idea for estimating the number of distinct elements in the stream. Suppose h is an idealized random hash function that maps $[1..n]$ to the interval $[0, 1]$. Suppose there are d distinct elements in the stream $\sigma = a_1, a_2, \ldots, a_m$. If h behaves like a random function then the set $\{h(a_1), \ldots, h(a_m)\}$ will behave like a collection of d independent uniformly distributed random variables in $[0, 1]$. Let $\theta = \min\{h(a_1), \ldots, h(a_m)\}$; the expectation of θ is $\frac{1}{d+1}$ and hence $1/\theta$ is good estimator. In the stream setting we can compute θ by hashing each incoming value and keeping track of the minimum. We only need to have one number in memory. Although simple, the algorithm assumes idealized hash functions and we only have an unbiased estimator. To convert the idea to an implementable algorithm with proper guarantees requires work. There are several papers on this problem and we will now discuss some of the approaches.

3.1 The AMS algorithm

Here we describe an algorithm with better parameters but it only gives a constant factor approximation. This is due to Alon, Matias and Szegedy in their famous paper [1] on estimating frequency moments. We need some notation. For an integer $t > 0$ let zeros(t) denote the number of zeros that the binary representation of t ends in; equivalenty

$$\text{zeros}(t) = \max\{i :\mid 2^i \text{ divides } t\}.$$

```
AMS-DistinctElements:

$\mathcal{H}$ is a 2-universal hash family from $[n]$ to $[n]$
choose $h$ at random from $\mathcal{H}$
$z \leftarrow 0$
While (stream is not empty) do
   $a_i$ is current item
   $z \leftarrow \max\{z, \text{zeros}(h(a_i))\}$
endWhile
Output $2^{z+\frac{1}{2}}$
```

First, we note that the space and time per element are $O(\log n)$. We now analyze the quality of the approximation provided by the output. Recall that $h(a_j)$ is uniformly distributed in $[n]$. We will assume for simplicity that n is a power of 2.

Let d to denote the number of distinct elements in the stream and let them be $b_1, b_2, \ldots, b_d$. For a given r let $X_{r,j}$ be the indicator random variable that is 1 if zeros$(h(b_j)) \geq r$. Let $Y_r = \sum_j X_{r,j}$. That is, Y_r is the number of distinct elements whose hash values have atleast r zeros.

Since $h(b_j)$ is uniformaly distribute in $[n]$,

$$\mathbf{E}[X_{r,j}] = \Pr[\text{zeros}(h(b_j)) \geq r] = \frac{(n/2^r)}{n} \geq \frac{1}{2^r}.$$

Therefore

$$\mathbf{E}[Y_r] = \sum_j \mathbf{E}[X_{r,j}] = \frac{d}{2^r}.$$

Thus we have $\mathbf{E}[Y_{\log d}] = 1$ (assuming d is a power of 2).

Now we compute the variance of Y_r. Note that the variables $X_{r,j}$ and $X_{r,j'}$ are pairwise independent since $\mathcal{H}$ is 2-universal. Hence

$$\mathbf{Var}[Y_r] = \sum_j \mathbf{Var}[X_{r,j}] \leq \sum_j \mathbf{E}[X_{r,j}^2] = \sum_j \mathbf{E}[X_{r,j}] = \frac{d}{2^r}.$$

Using Markov's inequality

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \mathbf{E}[Y_r] \leq \frac{d}{2^r}.$$

Using Chebyshev's inequality

$$\Pr[Y_r = 0] = \Pr[|Y_r - \mathbf{E}[Y_r]| \geq \frac{d}{2^r}] \leq \frac{\mathbf{Var}[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}.$$

Let z' be the value of z at the end of the stream and let $d' = 2^{z'+\frac{1}{2}}$ be the estimate for d output by the algorithm. We claim that d' cannot be too large compared to d with constant probability. Let a be the smallest integer such that $2^{a+\frac{1}{2}} \geq 3d$.

$$\Pr[d' \geq 3d] = \Pr[Y_a > 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

Now we claim that d' is not too small compared to d with constant probability. For this let b the largest integer such that $2^{b+\frac{1}{2}} \leq d/3$. Then,

$$\Pr[d' \leq d/3] = \Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

Thus, the algorithm provides $(1/3, \sqrt{2}/3 \simeq 0.4714)$-approximation to the number of distinct elements. Using the median trick we can make the probability of success be at least $(1-\delta)$ to obtain a $(1/3, \delta)$-approximation by running $O(\log \frac{1}{\delta})$-parallel and independent copies of the algorithm. The time and space will be $O(\log \frac{1}{\delta} \log n)$.

3.2 A $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon^2} \log n)$ space

Bar-Yossef et al. [2] described three algorithms for distinct elements, that last of which gives a bound $\tilde{O}(\epsilon^2 + \log n))$ space and amortized time per element $O(\log n + \log \frac{1}{\epsilon})$; the notation $\tilde{O})$ suppresses dependence on $\log \log n$ and $\log 1/\epsilon$. Here we describe their first algorithm that gives gives an (ϵ, δ_0)-approximation in space $O(\frac{1}{\epsilon^2} \log n)$ and $O(\log 1/\epsilon \log n)$ time per update; via the median trick, with an additional $\log 1/\delta$ factor, we can obtain a (ϵ, δ)-approximation.

The algorithm is based on the Flajolet-Martin idea of hashing to $[0, 1]$ and taking the minimum but with a small and important technical tweak. Let $t = \frac{c}{\epsilon^2}$ for some constant c to be fixed later.

```
BJKST-DistinctElements:

$\mathcal{H}$ is a 2-universal hash family from $[n]$ to $[N = n^3]$
choose $h$ at random from $\mathcal{H}$
$t \leftarrow \frac{c}{\epsilon^2}$
While (stream is not empty) do
    $a_i$ is current item
    Update the smallest $t$ hash values seen so far with $h(a_i)$
endWhile
Let $v$ be the $t$'th smallest value seen in the hast values.
Output $tN/v$.
```

We observe that the algorithm can be implemented by keeping track of t hash values each of which take $O(\log n)$ space and hence the space usage is $O(t \log n) = O(\frac{1}{\epsilon^2} \log n)$. The t values can be stored in a binary search tree so that when a new item is processed we can update the data structure in $O(\log t)$ searches which takes total time $O(\log \frac{1}{\epsilon} \log n)$.

The intution of the algorithm is as follows. As before let d be the number of distinct elements and let them be $b_1, \ldots, b_d$. The hash values $h(b_1), \ldots, h(b_d)$ are uniformly distributed in $[0, 1]$. For any fixed t we expect about t/d hash values to fall in the interval $[0, t/d]$ and the t'th smallest hash value v should be around t/d and this justifies the estimator for d being t/v. Now we formally analyse the properties of this estimator.

We chose the hash family to map $[n]$ to $[n^3]$ and therefore with probability at least $(1 - 1/n)$ the random hash function h is injective over $[n]$. We will assume this is indeed the case. Moreover we will assume that $n \geq \sqrt{\epsilon/24}$ which implies in particular that $\epsilon t/(4d) \geq 1/N$. Let d' the estimate returned by the algorithm.

Lemma 2 $\Pr[d' < (1 - \epsilon)d] \leq 1/6]$.

Proof: The values $h(b_1), \ldots, h(b_d)$ are uniformly distributed in $1..N$. If $d' < (1 - \epsilon)d$ it implies that $v > \frac{tN}{(1-\epsilon)d}$; that is less than t values fell in the interval $[1, \frac{tN}{(1-\epsilon)d}]$. Let X_i be the indicator random variable for the event that $h(b_i) \leq (1 + \epsilon)tN/d$ and let $Y = \sum_{i=}^{d} X_i$.

Since $h(b_i)$ is distributed uniformly in $[1..N]$, taking rounding errors into account, we have $(1 + \epsilon/2)t/d \leq \mathbf{E}[X_i] \leq (1 + 3\epsilon/2)t/d$ and hence $\mathbf{E}[Y] \geq t(1 + \epsilon/2)$. We have $\mathbf{Var}[Y] \leq \mathbf{E}[Y] \leq t(1 + 3\epsilon/2)$ (due to pairwise independence, Lemma 1)). We have $d' < (1 - \epsilon)d$ only if $Y < t$ and by Chebyshev,

$$\Pr[Y < t] \leq \Pr[|Y - \mathbf{E}[Y]| \geq \epsilon t/2] \leq \frac{4\mathbf{Var}[Y]}{\epsilon^2 t^2} \leq 12(\epsilon^2 t^2) \leq 1/6.$$

□

Lemma 3 $\Pr[d' > (1 + \epsilon)d] \leq 1/6]$.

Proof: Suppose $d' > (1 + \epsilon)d$, that is $v < \frac{tN}{(1+\epsilon)d}$. This implies that more than t hash values are less than $\frac{tN}{(1+\epsilon)d} \leq (1 - \epsilon/2)tN/d$. We will show that this happens with small probability.

Let X_i be the indicator random variable for the event $h(b_i) < (1 - \epsilon/2)tN/d$ and let $Y = \sum_{i=1}^{d} X_i$. We have $\mathbf{E}[X_i] < (1 - \epsilon/2)t/d + 1/N \leq (1 - \epsilon/4)t/d$ (the $1/N$ is for rounding errors). Hence $\mathbf{E}[Y] \leq (1 - \epsilon/4)t$. As we argued $d' > (1 + \epsilon)d$ happens only if $Y > t$. By Cheybyshev,

$$\Pr[Y > t] \leq \Pr[|Y - \mathbf{E}[Y]| \geq \epsilon t/4] \leq \frac{16\mathbf{Var}[Y]}{\epsilon^r t^2} \leq 16/(\epsilon^2 t^2) \leq 1/6.$$

□

Bibliographic Notes: Our description of the AMS algorithm is taken from notes of Amit Chakrabarti. Flajolet and Martin proposed the basic hash function based algorithm in [5]. [2] Kane, Nelson and Woodruff [7] described an (ϵ, δ_0)-approximation for a fixed δ_0 in space $O(\frac{1}{\epsilon^2} + \log n)$ (the time per update is also the same). This is a theoretically optimum algorithm since there are lower bounds of $\Omega(\epsilon^2)$ and $\Omega(\log n)$ on the space required for an ϵ-approximation. We refer the reader to [4, 6] for analysis and empirical evaluation of an algorithm called HyperLogLog which seems to very well in practice.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. Preliminary version in *Proc. of ACM STOC* 1996.

[2] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[3] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. Preliminary version in *Proc. ACM STOC*, 1977.

[4] Philippe Flajolet, Eric Fusy, Olivier Gandouet, Frédéric Meunier, et al. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Analysis of Algorithms 2007 (AofA07)*, pages 127–146, 2007.

[5] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[6] Stefan Heule, Marc Nunkesser, and Alex Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the EDBT 2013 Conference*, Genoa, Italy, 2013.

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

CS 598CSC: Algorithms for Big Data	**Lecture date: Sept 2, 2014**
Instructor: Chandra Chekuri	**Scribe: Chandra Chekuri**

1 AMS Sampling

We have seen reservoir sampling and the related weighted sampling technique to obtain independent samples from a stream without the algorithm knowing the length of the stream. We now discuss a technique to sample from a stream $\sigma = a_1, a_2, \ldots, a_m$ where the tokens a_j are integers from $[n]$ and we wish to estimate a function

$$g(\sigma) := \sum_{i \in [n]} g(f_i)$$

where f_i is the frequency of i and g is a real-valued function such that $g(0) = 0$. A natural example is to estimate frequency moments $F_k = \sum_{i \in [n]} f_i^k$; here we have $g(x) = x^k$, a convex function for $k \geq 1$. Another example is the empirical entropy of σ defined as $\sum_{i \in [n]} p_i \log p_i$ where $p_i = \frac{f_i}{m}$ is the empirical probability of i; here $g(x) = x \log x$.[1]

AMS sampling from the famous paper [**?**] gives an unbiased estimator for $g(\sigma)$. The estimator is based on a random variable Y defined as follows. Let J be a uniformly random sample from $[m]$. Let $R = |\{j \mid a_j = a_J, J \leq j \leq m\}|$. That is, R is the count of the number of tokens after J that are for the same coordinate. Then, let Y the estimate defined as:

$$Y = m(g(R) - g(R-1)).$$

The lemma below shows that Y is an unbiased estimator of $g(\sigma)$.

Lemma 1

$$\mathbf{E}[Y] = g(\sigma) = \sum_{i \in [n]} g(f_i).$$

Proof: The probability that $a_J = i$ is exactly f_i/m since J is a uniform sample. Moreover if $a_J = i$ then R is distributed as a uniform random variable over $[f_i]$.

$$\begin{aligned}
\mathbf{E}[Y] &= \sum_{i \in [n]} \Pr[a_J = i]\mathbf{E}[Y | a_J = i] \\
&= \sum_{i \in [n]} \frac{f_i}{m}\mathbf{E}[Y | a_J = i] \\
&= \sum_{i \in [n]} \frac{f_i}{m} \sum_{\ell=1}^{f_i} m \frac{1}{f_i} \left(g(\ell) - g(\ell - 1)\right) \\
&= \sum_{i \in [n]} g(f_i).
\end{aligned}$$

□

One can estimate Y using small space in the streaming setting via the reservoir sampling idea for generating a uniform sample. The algorithm is given below; the count R gets reset whenever a new sample is picked.

[1]In the context of entropy, by convention, $x \log x = 0$ for $x = 0$.

```
AMSEstimate:
$s \leftarrow$ null
$m \leftarrow 0$
$R \leftarrow 0$
While (stream is not done)
   $m \leftarrow m + 1$
   $a_m$ is current item
   Toss a biased coin that is heads with probability $1/m$
   If (coin turns up heads)
         $s \leftarrow a_m$
         $R \leftarrow 1$
   Else If ($a_m == s$)
         $R \leftarrow R + 1$
endWhile
Output $m(g(R) - g(R-1))$
```

To obtain a (ϵ, δ)-approximation via the estimator Y we need to estimate $\mathbf{Var}[Y]$ and apply standard tools. We do this for frequency moments now.

1.1 Application to estimating frequency moments

We now apply the AMS sampling to estimate F_k the k'th frequency moment for $k \geq 1$. We have already seen that Y is an exact statistication estimator for F_k when we set $g(x) = x^k$. We now estimate the variance of Y in this setting.

Lemma 2 *When $g(x) = x^k$ and $k \geq 1$,*

$$\mathbf{Var}[Y] \leq kF_1F_{2k-1} \leq kn^{1-\frac{1}{k}}F_k^2.$$

Proof:

$$\begin{aligned}
\mathbf{Var}[Y] &\leq \mathbf{E}[Y^2] \\
&\leq \sum_{i\in[n]} \Pr[a_J = i] \sum_{\ell=1}^{f_i} \frac{m^2}{f_i} \left(\ell^k - (\ell-1)^k\right)^2 \\
&\leq \sum_{i\in[n]} \frac{f_i}{m} \sum_{\ell=1}^{f_i} \frac{m^2}{f_i} (\ell^k - (\ell-1)^k)(\ell^k - (\ell-1)^k) \\
&\leq m \sum_{i\in[n]} \sum_{\ell=1}^{f_i} k\ell^{k-1}(\ell^k - (\ell-1)^k) \qquad \text{(using } (x^k - (x-1)^k) \leq kx^{k-1}\text{)} \\
&\leq km \sum_{i\in[n]} f_i^{k-1} f_i^k \\
&\leq kmF_{2k-1} = kF_1F_{2k-1}.
\end{aligned}$$

We now use convexity of the function x^k for $k \geq 1$ to prove the second part. Note that $\max_i f_i = F_\infty$.

$$F_1F_{2k-1} = (\sum_i f_i)(\sum_i f_i^{2k-1}) \leq (\sum_i f_i)F_\infty^{k-1}(\sum_i f_i^k) \leq (\sum_i f_i)(\sum_i f_i^k)^{\frac{k-1}{k}}(\sum_i f_i^k).$$

Using the preceding inequality, and the inequality $(\sum_{i=1}^{n} x_i)/n \leq ((\sum_{i=1}^{n} x_i^k)/n)^{\frac{1}{k}}$ for all $k \geq 1$ (due to the convexity of the function $g(x) = x^k$), we obtain that

$$F_1 F_{2k-1} \leq (\sum_i f_i)(\sum_i f_i^k)^{\frac{k-1}{k}}(\sum_i f_i^k) \leq n^{1-1/k}(\sum_i f_i^k)^{\frac{1}{k}}(\sum_i f_i^k)^{\frac{k-1}{k}}(\sum_i f_i^k) \leq n^{1-1/k}(\sum_i f_i^k)^2.$$

□

Thus we have $\mathbf{E}[Y] = F_k$ and $\mathbf{Var}[Y] \leq kn^{1-1/k}F_k^2$. We now apply the trick of reducing the variance and then the median trick to obtain a high-probability bound. If we take h independent estimators for Y and take their average the variance goes down by a factor of h. We let $h = \frac{c}{\epsilon^2}kn^{1-1/k}$ for some fixed constant c. Let Y' be the resulting averaged estimator. We have $\mathbf{E}[Y'] = F_k$ and $\mathbf{Var}[Y'] \leq \mathbf{Var}[Y]/h \leq \frac{\epsilon^2}{c}F_k^2$. Now, using Chebyshev, we have

$$\Pr[|Y' - \mathbf{E}[Y']| \geq \epsilon \mathbf{E}[Y']] \leq \mathbf{Var}[Y']/(\epsilon^2 \mathbf{E}[Y']^2) \leq \frac{1}{c}.$$

We can choose $c = 3$ to obtain a $(\epsilon, 1/3)$-approximation. By using the median trick with $\Theta(\log \frac{1}{\delta})$ independent estimators we can obtain a (ϵ, δ)-approximation. The overall number of estimators we run independently is $O(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2} \cdot n^{1-1/k})$. Each estimator requires $O(\log n + \log m)$ space since we keep track of one index from $[m]$, one count from $[m]$, and one item from $[n]$. Thus the space usage to obtain a (ϵ, δ)-approximation is $O(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2} \cdot n^{1-1/k} \cdot (\log m + \log n))$. The time to process each stream element is also the same.

The space complexity of $\tilde{O}(n^{1-1/k})$ is not optimal for estimating F_k. One can achieve $\tilde{O}(n^{1-2/k})$ which is optimal for $k > 2$ and one can in fact achieve poly-logarithmic space for $1 \leq k \leq 2$. We will see these results later in the course.

Bibliographic Notes: See Chapter 1 of the draft book by McGregor and Muthukrishnan; see the application of AMS sampling for estimating the entropy. See Chapter 5 of Amit Chakrabarti for the special case of frequency moments explained in detail. In particular he states a clean lemma that bundles the variance reduction technique and the median trick.

CS 598CSC: Algorithms for Big Data	**Lecture date: Sept 2, 2014**
Instructor: Chandra Chekuri	**Scribe: Chandra Chekuri**

1 F_2 Estimation

We have seen a generic algorithm for estimating the F_k, the k'th frequency moment of a stream using $\tilde{O}(n^{1-1/k})$-space for $k \geq 1$. Now we will see an amazingly simple algorithm for F_2 estimation due to [2].

```
AMS-$F_2$-Estimate:

$\mathcal{H}$ is a 4-universal hash family from $[n]$ to $\{-1,1\}$
choose $h$ at random from $\mathcal{H}$
$z \leftarrow 0$
While (stream is not empty) do
   $a_j$ is current item
   $z \leftarrow z + h(a_j)$
endWhile
Output $z^2$
```

An conceptually equivalent way to describe the algorithm is the following.

```
AMS-$F_2$-Estimate:

Let $Y_1, Y_2, \ldots, Y_n$ be $\{-1,+1\}$ random variable that are 4-wise independent
$z \leftarrow 0$
While (stream is not empty) do
   $a_j$ is current item
   $z \leftarrow z + Y_{a_j}$
endWhile
Output $z^2$
```

The difference between the two is that the former one is a streaming friendly. Instead of keeping $Y_1, \ldots, Y_n$ explicity we sample h from a 4-wise independent hash family so that h can be stored compactly in $O(\log n)$-space and we can generate $Y_i = h(i)$ in $O(\log n)$ time on the fly. We will analyze the algorithm in the second description.

Let $Z = \sum_{i \in [n]} f_i Y_i$ be the random variable that represents the value of z at the end of the stream. Note that for all $i \in [n]$, $\mathbf{E}[Y_i] = 0$ and $\mathbf{E}[Y_i^2] = 1$. Moreover, since $Y_1, \ldots, Y_n$ are 4-wise-independent and hence also 2-wise independent, $\mathbf{E}[Y_i Y_{i'}] = 0$ for $i \neq i'$. The expected value of the output is

$$\mathbf{E}[Z^2] = \sum_{i,i' \in [n]} f_i f_{i'} \mathbf{E}[Y_i Y_{i'}] = \sum_{i \in [n]} f_i^2 \mathbf{E}[Y_i^2] + \sum_{i \neq i'} f_i f_{i'} \mathbf{E}[Y_i Y_{i'}] = \sum_{i \in [n]} f_i^2 = F_2.$$

We can also compute the variance of the output which is $\mathbf{E}[Z^4]$.

$$\mathbf{E}[Z^4] = \sum_{i \in [n]} \sum_{j \in [n]} \sum_{k \in [n]} \sum_{\ell \in [n]} f_i f_j f_k f_\ell \mathbf{E}[Y_i Y_j Y_k Y_\ell].$$

Via the 4-wise independence of the Y's we have that $\mathbf{E}[Y_i Y_j Y_k Y_\ell] = 0$ if there is an index among i, j, k, ℓ that occurs exactly once in the multiset, otherwise it is 1. If it is 1 there are two cases: all indices are the same or there are two distinct indices that occur twice each. Therefore,

$$\mathbf{E}[Z^4] = \sum_{i\in[n]}\sum_{j\in[n]}\sum_{k\in[n]}\sum_{\ell\in[n]} f_i f_j f_k f_\ell \mathbf{E}[Y_i Y_j Y_k Y_\ell] = \sum_{i\in[n]} f_i^4 + 6\sum_{i=1}^{n}\sum_{j=i+1}^{n} f_i^2 f_j^2.$$

Thus, we have

$$\begin{aligned}
\mathbf{Var}[Z^2] &= \mathbf{E}[Z^4] - (\mathbf{E}[Z^2])^2 \\
&= F_4 - F_2^2 + 6\sum_{i=1}^{n}\sum_{j=i+1}^{n} f_i^2 f_j^2 \\
&= F_4 - (F_4 + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} f_i^2 f_j^2) + 6\sum_{i=1}^{n}\sum_{j=i+1}^{n} f_i^2 f_j^2 \\
&= 4\sum_{i=1}^{n}\sum_{j=i+1}^{n} f_i^2 f_j^2 \\
&\leq 2F_2^2.
\end{aligned}$$

Let $X = Z^2$ be the output estimate. We have $\mathbf{E}[X] = F_2$ and $\mathbf{Var}[X] \leq 2F_2^2 \leq 2\mathbf{E}[X]^2$. We now apply the standard idea of averaging $O(1/\epsilon^2)$ estimates to reduce variance, apply Chebyshev on the average estimator to see that it is a ϵ-approximation with $> 1/2$ probability. Then we apply the median trick with $\log(\frac{1}{\delta})$-independent averaged estimators to obtain an (ϵ, δ)-approximation. The overall space requirement is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log n)$ and this is also the time to process each element.

2 (Linear) Sketching and Streaming with Updates

The F_2 estimation algorithm is amazingly simple and has the following interesting properties. Suppose σ_1 and σ_2 are two streams and the algorithm computes z_1 and z_2 on σ_1 and σ_2 It is easy to see that the algorithm on $\sigma = \sigma_1 \cdot \sigma_2$ (the concatenation of the two streams) computes $z_1 + z_2$. Thus the algorithm retains z as a *sketch* of the stream σ. Note that the output of the algorithm is not z but some function of z (in the case of F_2 estimation it is z^2). Moreover, in this special case the sketch is a *linear* sketch which we will define more formally later.

Formally a sketch of a stream σ is a data structure $z(\sigma)$ that has the property that if $\sigma = \sigma \cdot \sigma_2$, $z(\sigma)$ can be computed by combining the sketches $z(\sigma_1)$ and $z(\sigma_2)$. Ideally the combining algorithm should take small space as well. Note that the algorithm can post-process the sketch to output the estimator.

The power of sketching algorithms is illustrated by thinking of more general streaming models than what we have seen so far. We have considered streams of the form $a_1, a_2, \ldots, a_m$ where each a_i is a token, in particular an integer from $[n]$. Now we will consider the following model. We start with a n-dimensional vector/signal $\mathbf{x} = (0, 0, \ldots, 0)$ and the stream tokens consists of *updates* to coordinates of $\mathbf{x}$. Thus each token $a_t = (i_t, \Delta_t)$ where $i_t \in [n]$ and Δ_t is a number (could be a real number and be negative). The token a_t udpates the i'th coordinate of $\mathbf{x}$:

$$x_{i_t} \leftarrow x_{i_t} + \Delta_t.$$

We will let $\mathbf{x}_t$ be the value of $\mathbf{x}$ after the updates corresponding to $a_1, a_2, \ldots, a_t$.

If the Δ_t's are allowed to be negative the model is called *turnstile streams*; note that Δ_t being negative allows items to be deleted. If $\mathbf{x}_t$ is required to be always non-negative, we have the *strict* turnstile stream model. A further special case is when Δ_t is required to be positive and is called the *cash register* model.

Linear sketches are particularly simple and yet very powerful. A linear sketch corresponds to a $k \times n$ projection matrix M and the sketch for vector $\mathbf{x}$ is simply $M\mathbf{x}$. Composing linear sketches corresponds to simply addding the sketches since $M\mathbf{x} + M\mathbf{x}' = M(\mathbf{x} + \mathbf{x}')$. In the streaming setting when we see a token $a_t = (i_t, \Delta_t)$, updating the sketch corresponds to adding $\Delta_t M e_{i_t}$ to the sketch where e_{i_t} is the vector with 1 in row i_t and 0 every where else. To implement the algorithm in small space it would suffice to be able to generate the i'th column of M efficienty on the fly rather than storing the entire matrix M.

F_2 estimation as linear sketching: It is not hard to see that the F_2 estimation algorithm we have seen is essentially a linear sketch algorithm. Consider the matrix M with $k = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ rows where each entry is in $\{-1, 1\}$. The sketch is simply $\mathbf{z} = M\mathbf{x}$. The algorithm post-processes the sketch to output its estimator.

Note that because the sketch is linear it does not matter whether $\mathbf{x}$ is negative. In fact it is easy to see this from the analysis as well. In particular this implies that we can estimate $||f_\sigma - f_{\sigma'}||_2$ where f_σ and $f_{\sigma'}$ are the frequency vectors of σ and σ' respectively. Similarly, if $\mathbf{x}$, $\mathbf{x}'$ are two n-dimensional signals representing a time-series then the ℓ_2 norm of their difference can be estimated by making one pass of the signals even when the signals are given via a sequence of updates which can even be interspersed (of course we need to know the identity of the signals from which the updates are coming from).

3 Johnson-Lindenstrauss Lemma and Dimension Reduction in ℓ_2

The AMS linear sketch for F_2 estimation appears magical. One way to understand this is via the dimensionality reduction for ℓ_2 spaces given by the well-known Johnson-Lindenstrauss lemma which has many applications. The JL Lemma can be stated as follows.

Theorem 1 (JL Lemma) *Let Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be any n points/vectors in $\mathbb{R}^d$. For any $\epsilon \in (0, 1/2)$, there is linear map $f : \mathbb{R}^d \to \mathbb{R}^k$ where $k \leq 8 \ln n / \epsilon^2$ such that for all $1 \leq i < j \leq n$,*

$$(1 - \epsilon)||v_i - v_j||_2 \leq ||f(v_i) - f(v_j)||_2 \leq ||v_i - v_j||_2.$$

Moreover f can be obtained in randomized polynomial-time.

The implication of the JL Lemma is that any n points in d-dimensional Euclidean space can be projected to $O(\ln n/\epsilon^2)$-dimensions while preserving all their pairwise Euclidean distances.

The simple randomized algorithm that proves the JL Lemma is the following. Let M be a $k \times d$ matrix where each entry M_{ij} is picked independently from the standard $\mathcal{N}(0, 1)$ normal distribution. Then the map f is givens as $f(\mathbf{v}) = \frac{1}{\sqrt{k}} M\mathbf{v}$. We now sketch why this works.

Lemma 2 *Let $Z_1, Z_2, \ldots, Z_k$ be independent $\mathcal{N}(0, 1)$ random variables and let $Y = \sum_i Z_i^2$. Then, for $\epsilon \in (0, 1/2)$, there is a constant c such that,*

$$\Pr[(1 - \epsilon)^2 k \leq Y \leq (1 + \epsilon)^2 k] \leq 2e^{c\epsilon^2 k}.$$

In other words the sum of squares of k standard normal variables is sharply concentrated around its mean which is k. In fact the distribution of Y has a name, the χ^2 distribution with parameter k. We will not prove the preceding lemma. A proof via the standard Chernoff-type argument can be found in various places.

Assuming the lemma we can prove the following.

Lemma 3 *Let $\epsilon in (0, 1/2)$ and $\mathbf{v}$ be a unit-vector in $\mathbb{R}^d$, then $(1-\epsilon) \leq ||M\mathbf{v}||_2 \leq (1+\epsilon)||$ with probability at least $(1 - 2e^{c\epsilon^2 k})$.*

Proof: First we observe a well-known fact about normal distributions. Let X and Y be independent $\mathcal{N}(0,1)$ random variables. Then $aX + bY$ is $\mathcal{N}(0, \sqrt{a^2+b^2})$ random variable.

Let $\mathbf{u} = \sqrt{k}M\mathbf{v}$. Note that $\mathbf{u}$ is a random vector. Note that $u_i = \sum_{j=1}^n v_j \sqrt{k} M_{ij}$. Since each $\sqrt{k}M_{ij}$ is $\mathcal{N}(0,1)$ random variable and all entries are independent we have that $u_i \simeq \mathcal{N}(0,1)$ because the variance of u_i is $\sum_j v_i^2 = 1$ (note that $\mathbf{v}$ is a unit vector). Thus $u_1, u_2, \ldots, u_k$ are independent $\mathcal{N}(0,1)$ random variables. Therefore $||u||_2^2 = \sum_i =^k u_i^2$. Applying Lemma 2, we have

$$\Pr[(1-\epsilon)^2 k \leq ||u||_2^2 \leq (1+\epsilon)^2 k] \geq 1 - 2e^{c\epsilon^2 k}.$$

□

Unit-vectors are convenient for the proof but by scaling one obtains the following easy corollary.

Corollary 4 *Let $\epsilon in (0, 1/2)$ and $\mathbf{v}$ be any vector in $\mathbb{R}^d$, then $(1-\epsilon)||\mathbf{v}||_2 \leq ||M\mathbf{v}||_2 \leq (1+\epsilon)||||\mathbf{v}||_2$ with probability at least $(1 - 2e^{c\epsilon^2 k})$.*

Now the JL Lemma follows easisly via a union bound. Let $k = c' \ln n/\epsilon^2$ where c' is chosen based on c. Consider any pair $\mathbf{v}_i, \mathbf{v}_j$.

$$\Pr[(1-\epsilon)||\mathbf{v}_i - \mathbf{v}_j||_2 \leq ||M(\mathbf{v}_i - \mathbf{v}_j||_2 \leq (1+\epsilon)||\mathbf{v}_i - \mathbf{v}_j||_2] \geq (1 - 2e^{c\epsilon^2 k}) \geq 1 - 2e^{c\epsilon^2 \cdot c' \ln n/\epsilon^2} \geq 1 - \frac{2}{n^{cc'}}.$$

If $cc' \geq 3$ then the probability of the distance between $\mathbf{v}_i$ and $\mathbf{v}_j$ being preserved to within a relative ϵ-approximation is at least $1 - 1/n^3$. Since there are only $n(n-1)/2$ pairs of distances, the probability that all of them will be preserved to this error tolerance is, via the union bound, at least $(1 - 1/n)$.

Bibliographic Notes: See Chapter 6 of Amit Chakrabarti's lecture notes. A lot of work has been done in the algorithmic community to make the dimensionality reduction faster to evaluate. An interesting result is due to Achlioptas [1] who showed that the matrix M whose entries we chose from $\mathcal{N}(0,1)$ can in fact be chosen from $\{-1, 0, 1\}$; the discrete entries create a sparser matrix and the resulting matrix multiplication is computationally easier.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. Preliminary version in *Proc. of ACM STOC* 1996.

CS 598CSC: Algorithms for Big Data — **Lecture date: Sept 9, 2014**
Instructor: Chandra Chekuri — **Scribe: Chandra Chekuri**

1 Sketch for F_p Estimation when $0 < p \leq 2$

We have seen a linear sketching estimate for F_2 estimation that uses $O(\log n)$ space. Indyk [1] obtained a technically sophisticated and interesting sketch for F_k estimation where $0 < p \leq 2$ (note that p can be a real number) which uses polylog(n) space. Since the details are rather technical we will only give the high-level approach and refer the reader to the paper and related notes for more details. Note that for $p > 2$ there is a lower bound of $\Omega(n^{1-2/p})$ on the space required.

To describe the sketch for $0 < p \leq 2$ we will revisit the F_2 estimate via the JL Lemma approach that uses properties of the normal distribution.

```
F_2-ESTIMATE:

Let $Y_1, Y_2, \ldots, Y_n$ be sampled independenty from the $\mathcal{N}(0,1)$ distribution
$z \leftarrow 0$
While (stream is not empty) do
   $(i_j, \Delta_j)$ is current token
   $z \leftarrow z + \Delta_j \cdot Y_{i_j}$
endWhile
Output $z^2$
```

Let $Z = \sum_{i \in [n]} x_i Y_i$ be the random variable that represents the value of z at the end of the stream. The variable Z is a sum of independent normal variables and by the properties of the normal distribution $Z \sim \sqrt{\sum_i x_i^2} \cdot \mathcal{N}(0,1)$. Normal distribution is called 2-stable for this reason. More generally a distribution $\mathcal{D}$ is said to be p-stable if the following property holds: Let $Z_1, Z_2, \ldots, Z_n$ be independent random variables distributed according to $\mathcal{D}$. Then $\sum_i x_i Z_i$ has the same distribution as $\|x\|_p Z$ where $Z \sim \mathcal{D}$. Note that a p-stable distribution will be symmetric around 0.

It is known that p-stable distributions exist for all $p \in (0,2]$ and not for any $p > 2$. The p-stable distributions do not have, in general, an analytical formula except in some cases. We have already seen that the standard normal distribution is 2-stable. The 1-stable distribution is the Cauchy distribution which is the distribution of the ratio of two independent standard normal random variables. The density function of the Cauchy distribution is known to be $\frac{1}{\pi(1+x^2)}$; note that the Cauchy distribution does not have a finite mean or variance. We use $\mathcal{D}_p$ to denote a p-stable distribution.

Although a general p-stable distribution does not have an analytical formula it is known that one can sample from $\mathcal{D}_p$. Chambers-Mallows-Stuck method is the following:

- Sample θ uniformly from $[-\pi/2, \pi/2]$.
- Sample r uniformly from $[0,1]$.
- Ouput
$$\frac{\sin(p\theta)}{(\cos\theta)^{1/p}} \left(\frac{\cos((1-p)\theta)}{\ln(1/r)}\right)^{(1-p)/p}.$$

We need one more definition.

Definition 1 *The median of a distribution $\mathcal{D}$ is θ if for $Y \sim \mathcal{D}$, $\Pr[Y \leq \mu] = 1/2$. If $\phi(x)$ is the probability density function of $\mathcal{D}$ then we have $\int_{-\infty}^{\mu} \phi(x)dx = 1/2$.*

Note that a median may not be uniquely defined for a distribution. The distribution $\mathcal{D}_p$ has a unique median and so we will use the terminology median($\mathcal{D}_p$) to denote this quantity. For a distribution $\mathcal{D}$ we will refer to $|\mathcal{D}|$ the distribution of the absolute value of a random variable drawn from $\mathcal{D}$. If $\phi(x)$ is the density function of $\mathcal{D}$ then the density function of $|\mathcal{D}|$ is given by ψ, where $\psi(x) = 2\phi(x)$ if $x \geq 0$ and $\psi(x) = 0$ if $x < 0$.

```
F_p-ESTIMATE:

$k \leftarrow \Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$
Let $M$ be a $k \times n$ matrix where each $M_{ij} \sim \mathcal{D}_p$
$\mathbf{y} \leftarrow M\mathbf{x}$
Output $Y \leftarrow \frac{\text{median}(|y_1|,|y_2|,\ldots,|y_k|)}{\text{median}(|\mathcal{D}_p|)}$
```

By the p-stability property we see that each $y_i \sim \|x\|_p Y$ where $Y \sim \mathcal{D}_p$. First, consider the case that $k = 1$. Then the output $|y_1|/\text{median}(|\mathcal{D}_p|)$ is distributed according to $c|\mathcal{D}_p|$ where $c = \|x\|_p/\text{median}(|\mathcal{D}_p|)$. It is not hard to verify that the median of this distribution is $\|x\|_p$. Thus, the algorithm take k samples from this distribution and ouputs as the estimator the *sample* median. The lemma below shows that the sample median has good concentration properties.

Lemma 1 *Let $\epsilon > 0$ and let $\mathcal{D}$ be a distribution with density function ϕ and a unique median $\mu > 0$. Suppose ϕ is absolutely continuous on $[(1-\epsilon)\mu, (1+\epsilon)\mu]$ and let $\alpha = \min\{\phi(x) \mid x \in [(1-\epsilon)\mu, (1+\epsilon)\mu]$. Let $Y = \text{median}(Y_1, Y_2, \ldots, Y_k)$ where $Y_1, \ldots, Y_k$ are independent samples from the distribution $\mathcal{D}$. Then*

$$\Pr[|Y - \mu| \geq \epsilon\mu] \leq 2e^{-\frac{2}{3}\epsilon^2\mu^2\alpha^2 k}.$$

We sketch the proof to upper bound $\Pr[Y \leq (1-\epsilon)\mu]$. The other direction is similar. Note that by the definition of the median, $\Pr[Y_j \leq \mu] = 1/2$. Hence

$$\Pr[Y_j \leq (1-\epsilon)\mu] = 1/2 - \int_{(1-\epsilon)\mu}^{\mu} \phi(x)dx.$$

Let $\gamma = \int_{(1-\epsilon)\mu}^{\mu} \phi(x)dx$. It is easy to see that $\gamma \geq \alpha\epsilon\mu$.

Let I_j be the indicator event for $Y_i \leq (1-\epsilon)\mu$; we have $\mathbf{E}[I_j] = \Pr[Y_i \leq (1-\epsilon)\mu] \leq 1/2 - \gamma$. Let $I = \sum_j I_j$; we have $\mathbf{E}[I] = k(1/2 - \gamma)$. Since Y is the median of $Y_1, Y_2, \ldots, Y_k$, $Y \leq (1-\epsilon)\mu$ only if more than $k/2$ of I_j are true which is the same as $\Pr[I > (1+\delta)\mathbf{E}[I]]$ where $1 + \delta = \frac{1}{1-2\gamma}$. Now, via Chernoff bounds, this probability is at most $e^{-\gamma^2 k/3}$ for sufficiently small γ.

We can now apply the lemma to the estimator output by the algorithm. We let ϕ be the distribution of $c|\mathcal{D}_p|$. Recall that the median of this distribution if $\|x\|_p$ and the output of the algorithm is the median of k indepenent samples from this distribution. Thus, from the lemma,

$$\Pr[|Y - \|x\|_p| \geq \epsilon\|x\|_p] \leq 2e^{-\epsilon^2 k\mu^2\alpha^2/3}.$$

Let ϕ' be the distribution of $|\mathcal{D}_{\surd}|$ and μ' be the median of ϕ'. Then it can be seen that $\mu\alpha = \mu'\alpha'$ where $\alpha' = \min\{\phi'(x) \mid (1-\epsilon)\mu' \leq (1+\epsilon)\mu'\}$. Thus $\mu'\alpha'$ depends only on $\mathcal{D}_p$ and ϵ. Letting this be $c_{p,\epsilon}$ we have,

$$\Pr[|Y - \|x\|_p| \geq \epsilon\|x\|_p] \leq 2e^{-\epsilon^2 k c_{p,\epsilon}^2/3} \leq (1-\delta),$$

provided $k = \Omega(c_{p,\epsilon} \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta})$.

Technical Issues: There are several technical issues that need to be addressed to obtain a proper algorithm from the preceding description. First, the algorithm as described requires one to store the entire matrix M which is too large for streaming applications. Second, the constant k depends on $c_{p,\epsilon}$ which is not explicitly known since $\mathcal{D}_p$ is not well-understood for general p. To obtain a streaming algorithm, the very high-level idea is to derandomize the algorithm via the use of pseudorandom generators for small-space due to Nisan. See [1] for more details.

2 Counting Frequent Items

We have seen various algorithm for estimating various F_p norms for $p \geq 0$. Note that F_0 corresponds to number of distinct elements. In the limit, as $p \to \infty$, ℓ_p norm of a vector $\mathbf{x}$ is the maximum of the absolute values of the entries of $\mathbf{x}$. Thus, we can define the F_∞ norm to corresponds to finding the maximum frequency in $\mathbf{x}$. More generally, we would like to find the frequent items in a stream which are also called "heavy hitters". In general, it is not feasible to estimate the heaviest frequency with limited space if it is too small relative to m.

2.1 Misra-Greis algorithm for frequent items

Suppose we have a stream $\sigma = a_1, a_2, \ldots, a_m$ where $a_j \in [n]$, the simple setting and we want to find all elements in $[n]$ such that $f_i > m/k$. Note that there can be at most k such elements. The simplest case is when $k = 2$ when we want to know whether there is a "majority" element. There is a simple deterministic algorithm that perhaps you have all seen for $k = 2$ in an algorithm class. The algorithm uses an associative array data structure of size k.

```
MisraGreis($k$):

$D$ is an empty associative array
While (stream is not empty) do
  $a_j$ is current item
  If ($a_j$ is in $keys(D)$)
      $D[a_j] \leftarrow D[a_j] + 1$
  Else if ($|keys(A)| < k - 1$) then
      $D[a_j] \leftarrow 1$
  Else
      for each $\ell \in keys(D)$ do
          $D[\ell] \leftarrow D[\ell] - 1$
      Remove elements from $D$ whose counter values are 0
 endWhile
  For each $i \in keys(D)$ set $\hat{f}_i = D[i]$
  For each $i \notin keys(D)$ set $\hat{f}_i = 0$
```

We leave the following as an exercise to the reader.

Lemma 2 *For each $i \in [n]$:*

$$f_i - \frac{m}{k} \leq \hat{f}_i \leq f_i.$$

The lemma implies that if $f_i > m/k$ then $i \in keys(D)$ at the end of the algorithm. Thus one can use a second-pass over the data to compute the exact f_i only for the k itmes in $keys(D)$. This gives an $O(kn)$ time two-pass algorithm for finding all items which have frequency at least m/k.

Bibliographic Notes: For more details on F_p estimation when $0 < p \leq 2$ see the original paper of Indyk [1], notes of Amit Chakrabarti (Chapter 7) and Lecture 4 of Jelani Nelson's course.

References

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

CS 598CSC: Algorithms for Big Data | **Lecture date: Sept 11, 2014**
Instructor: Chandra Chekuri | **Scribe: Chandra Chekuri**

The Misra-Greis deterministic counting guarantees that all items with frequency $> F_1/k$ can be found using $O(k)$ counters and an update time of $O(\log k)$. Setting $k = 1/\epsilon$ one can view the algorithm as providing an additive ϵF_1 approximation for each f_i. However, the algorithm does not provide a sketch. One advantage of linear sketching algorithms is the ability to handle deletions. We now discuss two sketching algorithms that have a found a number of applications. These sketches can be used to for estimating point queries: after seeing a stream σ over items in $[n]$ we would like to estimate f_i the frequency of $i \in [n]$. More generally, in the turnstile model, we would like to estimate x_i for a given $i \in [n]$. We can only guarantee the estimate with an *additive* error.

1 CountMin Sketch

We firt describe the simpler CountMin sketch. The sketch maintains several counters. The counters are best visualized as a rectangular array of width w and depth d. With each row i we have a hash function $h_i : [n] \to [w]$ that maps elements to one of w buckets.

```
CountMin-Sketch($w, d$):

$h_1, h_2, \ldots, h_d$ are pair-wise independent hash functions from $[n] \to [w]$.
While (stream is not empty) do
  $a_t = (i_t, \Delta_t)$ is current item
  for $\ell = 1$ to $d$ do
      $C[\ell, h_\ell(i_j)] \leftarrow C[\ell, h_\ell(i_j)] + \Delta_t$
endWhile
For $i \in [n]$ set $\tilde{x}_i = \min_{\ell=1}^{d} C[\ell, h_\ell(i)]$.
```

The counter $C[\ell, j]$ simply counts the sum of all x_i such that $h_\ell(i) = j$. That is,

$$C[\ell, j] = \sum_{i : h_\ell(i) = j} x_i.$$

Exercise: CountMin is a linear sketch. What are the entries of the projection matrix?

We will analyze the sketch in the strick turnstile model where $x_i \geq 0$ for all $i \in [n]$; note that Δ_t we be negative.

Lemma 1 *Let $d = \Omega(\log \frac{1}{\delta})$ and $w > \frac{2}{\epsilon}$. Then for any fixed $i \in [n]$, $x_i \leq \tilde{x}_i$ and*

$$\Pr[\tilde{x}_i \geq x_i + \epsilon \|\mathbf{x}\|_1] \leq \delta.$$

Proof: Fix $i \in [n]$. Let $Z_\ell = C[\ell, h_\ell(i)]$ be the value of the counter in row ℓ to which i is hashed to. We have

$$\mathbf{E}[Z_\ell] = x_i + \sum_{i' \neq i} \Pr[h_\ell(i') = h_\ell(i)] x_{i'} = x_i + \sum_{i' \neq i} \frac{1}{w} x_{i'} \leq x_i + \frac{\epsilon}{2} \|\mathbf{x}\|_1.$$

Note that we used pair-wise independence of h_ℓ to conclude that $\Pr[h_\ell(i') = h_\ell(i)] = 1/w$.

By Markov's inequality (here we are using non-negativity of $\mathbf{x}$),

$$\Pr[Z_\ell > x_i + \epsilon\|\mathbf{x}\|_1] \leq 1/2.$$

Thus

$$\Pr[\min_\ell Z_\ell > x_i + \epsilon\|\mathbf{x}\|_1] \leq 1/2^d \leq \delta.$$

□

Remark: By choosing $\delta = \Omega(\log n)$ we can ensure with probability at least $(1 - 1/\text{poly}(n))$ that $\tilde{x}_i - x_i \leq \epsilon\|\mathbf{x}\|_1$ for *all* $i \in [n]$.

Exercise: For general turnstile streams where $\mathbf{x}$ can have negative entries we can take the median of the counters. For this estimate you should be able to prove the following.

$$\Pr[|\tilde{x}_i - x_i| \geq 3\epsilon\|\mathbf{x}\|_1] \leq \delta^{1/4}.$$

2 Count Sketch

Now we discuss the closely related Count sketch which also maintains an array of counters parameterized by the width w and depth d.

```
COUNT-SKETCH($w, d$):

$h_1, h_2, \ldots, h_d$ are pair-wise independent hash functions from $[n] \to [w]$.
$g_1, g_2, \ldots, g_d$ are pair-wise independent hash functions from $[n] \to \{-1, 1\}$.
While (stream is not empty) do
   $a_t = (i_t, \Delta_t)$ is current item
   for $\ell = 1$ to $d$ do
        $C[\ell, h_\ell(i_j)] \leftarrow C[\ell, h_\ell(i_j)] + g(i_t)\Delta_t$
endWhile
For $i \in [n]$ set $\tilde{x}_i = \text{median}\{g_1(i)C[1, h_1(i)], g_2(i)C[2, h_2(i), \ldots, g_d(i)C[d, h_d(i)]\}$.
```

Exercise: CountMin is a linear sketch. What are the entries of the projection matrix?

Lemma 2 *Let $d \geq \log \frac{1}{\delta}$ and $w > \frac{3}{\epsilon^2}$. Then for any fixed $i \in [n]$, $\mathbf{E}[\tilde{x}_i] = x_i$ and*

$$\Pr[|\tilde{x}_i - x_i| \geq \epsilon\|\mathbf{x}\|_2] \leq \delta.$$

Proof: Fix an $i \in [n]$. Let $Z_\ell = g_\ell(i)C[\ell, h_\ell(i)]$. For $i' \in [n]$ let $Y_{i'}$ be the indicator random variable that is 1 if $h_\ell(i) = h_\ell(i')$; that is i and i' collide in h_ℓ. Note that $\mathbf{E}[Y_{i'}] = \mathbf{E}[Y_{i'}^2] = 1/w$ from the pairwise independence of h_ℓ. We have

$$Z_\ell = g_\ell(i)C[\ell, h_\ell(i)] = g_\ell(i)\sum_{i'} g_\ell(i')x_{i'}Y_{i'}$$

Therefore,

$$\mathbf{E}[Z_\ell] = x_i + \sum_{i' \neq i} \mathbf{E}[g_\ell(i)g_\ell(i')Y_{i'}]x_{i'} = x_i,$$

because $\mathbf{E}[g_\ell(i)g_\ell(i')] = 0$ for $i \neq i'$ from pairwise independence of g_ℓ and $Y_{i'}$ is independent of $g_\ell(i)$ and $g_\ell(i')$. Now we upper bound the variance of Z_ℓ.

$$\begin{aligned}
\mathbf{Var}[Z_\ell] &= \mathbf{E}\left[(\sum_{i'\neq i} g_\ell(i)g_\ell(i')Y_{i'}x_{i'})^2\right] \\
&= \mathbf{E}\left[\sum_{i'\neq i} x_{i'}^2 Y_{i'}^2 + \sum_{i'\neq i''} x_{i'}x_{i''}g_\ell(i')g_\ell(i'')Y_{i'}Y_{i''}\right] \\
&= \sum_{i'\neq i} x_{i'}^2 \mathbf{E}[Y_{i'}^2] \\
&\leq \|\mathbf{x}\|_2^2/w.
\end{aligned}$$

Using Chebyshev,

$$\Pr[|Z_\ell - x_i| \geq \epsilon\|\mathbf{x}\|_2] \leq \frac{\mathbf{Var}[Z_\ell]}{\epsilon^2\|\mathbf{x}\|_2^2} \leq \frac{1}{\epsilon^2 w} \leq 1/3.$$

Now, via the Chernoff bound,

$$\Pr\left[|\text{median}\{Z_1,\ldots,Z_d\} - x_i| \geq \epsilon\|\mathbf{x}\|_2\right] \leq e^{-cd} \leq \delta.$$

Thus choosind $d = O(\log n)$ and taking the median guarantees the desired bound with high probability. □

Remark: By choosing $\delta = \Omega(\log n)$ we can ensure with probability at least $(1 - 1/\text{poly}(n))$ that $|\tilde{x}_i - x_i| \leq \epsilon\|\mathbf{x}\|_2$ for *all* $i \in [n]$.

3 Applications

Count and CountMin sketches have found a number of applications. Note that they have a similar structure though the guarantees are different. Consider the problem of estimating frequency moments. Count sketch outputs an estimate $\tilde{f}_i$ for f_i with an additive error of $\epsilon\|\mathbf{f}\|_2$ while CountMin guarantees an additive error of $\epsilon\|\mathbf{f}\|_1$ which is always larger. CountMin provides a one-sided error when $\mathbf{x} \geq 0$ which has some benefits. CountMin uses $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$ counters while Count sketch uses $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ counters. Note that the Misra-Greis algorithm uses $O(1/\epsilon)$-counters.

3.1 Heavy Hitters

We will call an index i an α-HH (for heavy hitter) if $x_i \geq \alpha\|x\|_1$ where $\alpha \in (0, 1]$. We would like to find S_α, the set of all α-heavy hitters. We will relax this assumption to output S such that

$$S_\alpha \subseteq S \subseteq S_{\alpha-\epsilon}.$$

Here we will assume that $\alpha < \alpha$ for otherwise the approximation does not make sense.

Suppose we used CountMin sketch with $w = 2/\epsilon$ and $\delta = c/n$ for sufficiently large c. Then, as we saw, with probability at least $(1 - 1/\text{poly}(n))$, for all $i \in [n]$,

$$x_i \leq \tilde{x}_i \leq x_i + \epsilon\|\mathbf{x}\|_1.$$

Once the sketch is computed we can simply go over all i and add i to S if $\tilde{x}_i \geq \alpha\|\mathbf{x}\|_1$. It is easy to see that S is the desired set.

Unfortunately the computation of S is expensive. The sketch has $O(\frac{1}{\epsilon}\log n)$ counters and processing each i takes time proportional to the number of counters and hence the total time is $O(\frac{1}{\epsilon}n\log n)$ to output a set S of size $O(\frac{1}{\alpha})$. It turns that by keeping additional information in the sketch in a hierarchical fashion one can cut down the time to be proportional to $O(\frac{1}{\alpha}\text{polylog}(n)))$.

3.2 Range Queries

In several application the range $[n]$ corresponds to an actual total ordering of the items. For instance $[n]$ could represent the discretization of time and $\mathbf{x}$ corresponds to the signal. In databases $[n]$ could represent ordered numerical attributes such as age of a person, height, or salary. In such settings range queries are very useful. A range query is an interval of the form $[i, j]$ where $i, j \in [n]$ and $i \leq j$. The goal is to output $\sum_{i\leq \ell \leq j} x_i$. Note that there are $O(n^2)$ potential queries.

There is a simple trick to solve this using the sketches we have seen. An interval $[i, j]$ is a *dyadic* interval/range if $j - i + 1$ is 2^k and 2^k divides $i - 1$. Assume n is a power of 2. Then the dyadic intervals of length 1 are $[1, 1], [2, 2], \ldots, [n, n]$. Those of length 2 are $[1, 2], [3, 4], \ldots$ and of length 4 are $[1, 4], [5, 8], \ldots$.

Claim 3 *Every range $[i, j]$ can be expressed as a disjoint union of at most $2\log n$ dyadic ranges.*

Thus it suffices to maintain accurate point queries for the dyadic ranges. Note that there are at most $2n$ dyadic ranges. They fall into $O(\log n)$ groups based on length; the ranges for a given length partition the entire interval. We can keep a separate CountMin sketch for the $n/2^i$ dyadic intervals of length i ($i = 0$ corresponds to the sketch for point queries). Using these $O(\log n)$ CountMin sketches we can answer any range query with an additive error of $\epsilon\|\mathbf{x}\|_1$. Note that a range $[i, j]$ is expressed as the sum of $2\log n$ point queries each of which has an additive error. So ϵ' for the sketches has to be chosen to be $\epsilon/(2\log n)$ to ensure an additive error of $\epsilon\|\mathbf{x}\|_1$ for the range queries.

By choosing $d = O(\log n)$ the error probability for all point queries in all sketches will be at most $1/\text{poly}(n)$. This will guarantee that all range queries will be answered to within an additive $\epsilon\|\mathbf{x}\|_1$. The total space will be $O(\frac{1}{\epsilon}\log^3 n)$

3.3 Sparse Recovery

Let $\mathbf{x} \in \mathbb{R}^n$ be a vector. Can we approximate $\mathbf{x}$ by a sparse vector $\mathbf{z}$? By sparse we mean that $\mathbf{z}$ has at most k non-zero entries for some given k (this is the same as saying $\|\mathbf{z}\|_0 \leq k$). A reasonable way to model this is to ask for computing the error

$$\text{err}_p^k(\mathbf{x}) = \min_{\mathbf{z}:\|\mathbf{z}\|_0\leq k} \|\mathbf{x} - \mathbf{z}\|_p$$

for some p. A typical choice is $p = 2$. It is easy to see that the optimum $\mathbf{z}$ is obtained by restricting $\mathbf{x}$ to its k largest coordinates (in absolute value). The question we ask here is whether we can estimate $\text{err}_2^k(\mathbf{x})$ efficiently in a streaming fashion. For this we use the Count sketch. Recall that by choosing $w = 3/\epsilon^2$ and $d = \Theta(\log n)$ the sketch ensures that with high probability,

$$\forall i \in [n], \quad |\tilde{x}_i - x_i| \leq \epsilon\|\mathbf{x}\|_2.$$

One can in fact show a generalization.

Lemma 4 *Count-Sketch with $w = 3k/\epsilon$ and $d = O(\log n)$ ensures that*

$$\forall i \in [n], \quad |\tilde{x}_i - x_i| \leq \frac{\epsilon}{\sqrt{k}} err_2^k(\mathbf{x}).$$

Proof: Let $S = \{i_1, i_2, \ldots, i_k\}$ be the indices of the largest coordinates in $\mathbf{x}$ and let $\mathbf{x}'$ be obtained from $\mathbf{x}$ by setting entries of $\mathbf{x}$ to zero for indices in S. Note that $\mathrm{err}_2^k(\mathbf{x}) = \|\mathbf{x}'\|_2$. Fix a coordinate i. Consider row ℓ and let $Z_\ell = g_\ell(i) C[\ell, h_\ell(i)]$ as before. Let A_ℓ be the event that there exists an index $t \in S$ such that $h_\ell(i) = h_\ell(t)$; that is any "big" coordinate collides with i under h_ℓ. Note that $\Pr[A_\ell] \leq \sum_{t \in S} \Pr[h_\ell(i) = \Pr[h_\ell(t)] \leq |S|/w \leq \epsilon/3$ by pair-wise independence of h. Now we estimate

$$\begin{aligned}
\Pr[|Z_\ell - x_i| \geq \frac{\epsilon}{\sqrt{k}} \mathrm{err}_2^k(\mathbf{x})] &= \Pr[|Z_\ell - x_i| \geq \frac{\epsilon}{\sqrt{k}} \|\mathbf{x}'\|_2] \\
&= \Pr[A_\ell] \cdot \Pr[|Z_\ell - x_i| \geq \frac{\epsilon}{\sqrt{k}} \|\mathbf{x}'\|_2] + \Pr[|Z_\ell - x_i| \geq \frac{\epsilon}{\sqrt{k}} \|\mathbf{x}'\|_2 \mid \neg A_\ell] \\
&\leq \Pr[A_\ell] + 1/3 < 1/2.
\end{aligned}$$

□

Now let $\tilde{\mathbf{x}}$ be the approximation to $\mathbf{x}$ that is obtained from the sketch. We can take the k largest coordinates of $\tilde{\mathbf{x}}$ to form the vector $\mathbf{z}$ and output $\mathbf{z}$. We claim that this gives a good approximation to $\mathrm{err}_2^k(\mathbf{x})$. To see this we prove the following lemma.

Lemma 5 *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that*

$$\|\mathbf{x} - \mathbf{y}\|_\infty \leq \frac{\epsilon}{\sqrt{k}} err_2^k(\mathbf{x}).$$

Then,

$$\|\mathbf{x} - \mathbf{z}\|_2 \leq (1 + 5\epsilon) err_2^k(\mathbf{x}),$$

where $\mathbf{z}$ is the vector obtained as follows: $\mathbf{z}_i = \mathbf{y}_i$ for $i \in T$ where T is the set of k largest (in absolute value) indices of $\mathbf{y}$ and $\mathbf{z}_i = 0$ for $i \notin T$.

Proof: Let $t = \frac{1}{\sqrt{k}} \mathrm{err}_2^k(\mathbf{x})$ to help ease the notation. Let S be the index set of the largest coordinates of $\mathbf{x}$. We have,

$$(\mathrm{err}_2^k(\mathbf{x}))^2 = kt^2 = \sum_{i \in [n] \setminus S} x_i^2 = \sum_{i \in T \setminus S} x_i^2 + \sum_{i \in [n] \setminus (S \cup T)} x_i^2.$$

We write:

$$\begin{aligned}
\|\mathbf{x} - \mathbf{z}\|_2^2 &= \sum_{i \in T} |x_i - z_i|^2 + \sum_{i \in S \setminus T} |x_i - z_i|^2 + \sum_{i \in [n] \setminus (S \cup T)} x_i^2 \\
&= \sum_{i \in T} |x_i - y_i|^2 + \sum_{i \in S \setminus T} x_i^2 + \sum_{i \in [n] \setminus (S \cup T)} x_i^2.
\end{aligned}$$

We treat each term separately. The first one is easy to bound.

$$\sum_{i \in T} |x_i - y_i|^2 \leq \sum_{i \in T} \epsilon^2 t^2 \leq \epsilon^2 k t^2.$$

The third term is common to $\|\mathbf{x}-\mathbf{z}\|_2$ and $\mathrm{err}_2^k(\mathbf{x})$. The second term is the one to care about.

Note that S is set of k largest coordinates in $\mathbf{x}$ and T is set of k largest coordinates in $\mathbf{y}$. Thus $|S \setminus T| = |T \setminus S|$, say their cardinality is $\ell \geq 1$. Since $\mathbf{x}$ and $\mathbf{y}$ are close in ℓ_∞ norm (that is they are close in each coordinate) it must mean that the coordinates in $S \setminus T$ and $T \setminus S$ are roughly the same value in $\mathbf{x}$. More precisely let $a = \max_{i \in S \setminus T} |x_i|$ and $b = \min_{i \in T \setminus S} |x_i|$. We leave it as an exercise to the reader to argue that that $a \leq b + 2\epsilon t$ since $\|\mathbf{x}-\mathbf{y}\|_\infty \leq \epsilon t$.

Thus,

$$\sum_{i \in S \setminus T} x_i^2 \leq \ell a^2 \leq \ell(b+2\epsilon t)^2 \leq \ell b^2 + 4\epsilon k t b + 4k\epsilon^2 t^2.$$

But we have

$$\sum_{i \in T \setminus S} x_i^2 \geq \ell b^2.$$

Putting things together,

$$\begin{aligned}
\|\mathbf{x}-\mathbf{z}\|_2^2 &\leq \ell b^2 + 4\epsilon k t b + \sum_{i \in [n] \setminus (S \cup T)} x_i^2 + 5k\epsilon^2 t^2 \\
&\leq \sum_{i \in T \setminus S} x_i^2 + \sum_{i \in [n] \setminus (S \cup T)} x_i^2 + 4\epsilon(\mathrm{err}_2^k(\mathbf{x}))^2 + 5\epsilon^2(\mathrm{err}_2^k(\mathbf{x}))^2 \\
&\leq (\mathrm{err}_2^k(\mathbf{x}))^2 + 9\epsilon(\mathrm{err}_2^k(\mathbf{x}))^2.
\end{aligned}$$

The lemma follows by by the fact that for sufficiently small ϵ, $\sqrt{1+9\epsilon} \leq 1+5\epsilon$.

□

Bibliographic Notes: Count sketch is by Charikar, Chen and Farach-Colton [1]. CountMin sketch is due to Cormode and Muthukrishnan [4]; see the papers for several applications. Cormode's survey on sketching in [2] has a nice perspective. See [3] for a comparative analysis (theoretical and experimenta) of algorithms for frinding frequent items. A deterministic variant of CountMin called CR-Precis is interesting; see `http://polylogblog.wordpress.com/2009/09/22/bite-sized-streams-cr-precis/` for a blog post with pointers and some comments. The applications are taken from the first chapter in the draft book by McGregor and Muthukrishnan.

References

[1] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[2] Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[3] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *VLDB J.*, 19(1):3–20, 2010.

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

CS 598CSC: Algorithms for Big Data	**Lecture date: Sept 23, 2014**
Instructor: Chandra Chekuri	**Scribe: Chandra Chekuri**

1 Priority Sampling and Sum Queries

Suppose we have a stream $a_1, a_2, \ldots, a_n$ (yes, we are changing notation here from m to n for the length of the stream) of objects and each a_i has a *non-negative* weight w_i. We want to store a representative sample $S \subset [n]$ of the items so that we can answer subset sum queries. That is, given a query $I \subseteq [n]$ we would like to answer $\sum_{i \in I} w_i$. One way to do this as follows. Suppose we pick S as follows: sample each $i \in [n]$ independently with probability p_i and if i is chosen we set a scaled weight $\hat{w}_i = w_i/p_i$. Now, given a query I we output the estimate for its weight as $\sum_{i \in I \cap S} \hat{w}_i$. Note that expectation of the estimate is equal to $w(I)$. The disadvantage of this scheme is mainly related to the fact that we cannot control the size of sample. This means that we cannot fully utilize the memory available. Related to the first, is that if we do not know the length of stream apriori, we cannot figure out the sampling rate even if we are willing to be flexible in the size of the memory. An elegant scheme of Duffield, Lund and Thorup, called priority sampling overcomes these limitations. They considered the setting where we are given a parameter k for the size of the sample S and the goal is maintain a k-sample S along with some weights $\hat{w}_i$ for $i \in S$ such that we can answer subset sum queries.

Their scheme is the following, described as if $a_1, a_2, \ldots, a_n$ are available offline.

1. For each $i \in [n]$ set priority $q_i = w_i/u_i$ where u_i is chosen uniformly (and independently from other items) at random from $[0, 1]$.
2. S is the set of items with the k highest priorities.
3. τ is the $(k + 1)$'st highest priority. If $k \geq n$ we set $\tau = 0$.
4. If $i \in S$, set $\hat{w}_i = \max\{w_i, \tau\}$, else set $\hat{w}_i = 0$.

We observe that the above sampling can be implemented in the streaming setting by simply keeping the current sample S and current threshold τ. We leave it as an exercise to show that this informatino can be updated when a new item arrives.

We show some nice and non-obvious properties of priority sampling. We will assume for simplicity that $1 < k < n$. The first one is the basic one that we would want.

Lemma 1 $\mathbf{E}[\hat{w}_i] = w_i$.

Proof: Fix i. Let $A(\tau')$ be the event that the k'th highest priority among items $j \neq i$ is τ'. Note that $i \in S$ if $q_i = w_i/u_i \geq \tau'$ and if $i \in S$ then $\hat{w}_i = \max\{w_i, \tau'\}$, otherwise $\hat{w}_i = 0$. To evaluate $\Pr[i \in S \mid A(\tau')]$ we consider two cases.

Case 1: $w_i \geq \tau'$. Here we have $\Pr[i \in S \mid A(\tau')] = 1$ and $\hat{w}_i = w_i$.

Case 2: $w_i < \tau'$. Then $\Pr[i \in S \mid A(\tau')] = \frac{w_i}{\tau'}$ and $\hat{w}_i = \tau'$.

In both cases we see that $\mathbf{E}[\hat{w}_i] = w_i$. □

The previous claim shows that the estimator $\sum_{I \cap S} \hat{w}_i$ has expectation equal to $w(I)$. We can also estimate the variance of $\hat{w}_i$ via the threshold τ.

Lemma 2 $\mathbf{Var}[\hat{w}_i] = \mathbf{E}[\hat{v}_i]$ *where* $\hat{v}_i = \begin{cases} \tau \max\{0, \tau - w_i\} & \text{if } i \in S \\ 0 & \text{if } i \notin S \end{cases}$

Proof: Fix i. We define $A(\tau')$ to be the event that τ' is the k'th highest priority among elements $j \neq i$. The proof is based on showing that

$$\mathbf{E}[\hat{v}_i \mid A(\tau')] = \mathbf{E}[\hat{w}_i^2 \mid A(\tau')] - w_i^2.$$

From the proof outline of the preceding lemma, we estimate the lhs of as

$$\begin{aligned} \mathbf{E}[\hat{v}_i \mid A(\tau')] &= \Pr[i \in S \mid A(\tau')] \times \mathbf{E}[\hat{v}_i \mid i \in S \wedge A(\tau')] \\ &= \min\{1, w_i/\tau'\} \times \tau' \max\{0, \tau' - w_i\} \\ &= \max\{0, w_i\tau' - w_i^2\}. \end{aligned}$$

Now we analyze the rhs,

$$\begin{aligned} \mathbf{E}[\hat{w}_i^2 \mid A(\tau')] &= \Pr[i \in S \mid A(\tau')] \times \mathbf{E}[\hat{w}_i^2 \mid i \in S \wedge A(\tau')] \\ &= \min\{1, w_i/\tau'\} \times (\max\{w_i, \tau'\})^2 \\ &= \max\{w_i^2, w_i\tau'\}. \end{aligned}$$

□

Surprisingly, if $k \geq 2$ then the covariance between $\hat{w}_i$ and $\hat{w}_j$ for any $i \neq j$ is equal to 0.

Lemma 3 $\mathbf{E}[\hat{w}_i \hat{w}_j] = 0$.

In fact the previous lemma is a special case of a more general lemma below.

Lemma 4 $\mathbf{E}[\prod_{i \in I} \hat{w}_i] = \prod_{i=1}^{k} w_i$ *if* $|I| \leq k$ *and is* 0 *if* $|I| > k$.

Proof: It is easy to see that if $|I| > k$ the product is 0 since at least one of them is not in the sample. We now assume $|I| \leq k$ and prove the desired claim by inducion on $|I|$. In fact we need a stronger hypothesis. Let τ'' be the $(k - |I| + 1)$'th highest priority among the items $j \neq i$. We will condition on τ'' and prove that $\mathbf{E}[\prod_{i \in I} \hat{w}_i \mid A(\tau'')] = \prod_{i \in I} w_i$. For the base case we have seen the proof for $|I| = 1$.

Case 1: There is $h \in I$ such that $w_h > \tau''$. Then clearly $h \in S$ and $\hat{w}_h = w_h$. In this case

$$\mathbf{E}[\prod_{i \in I} \hat{w}_i \mid A(\tau'')] = w_h \cdot \mathbf{E}[\prod_{i \in I \setminus \{h\}} \hat{w}_i \mid A(\tau'')],$$

and we apply induction to $I \setminus \{h\}$. Technically the term $\mathbf{E}[\prod_{i \in I \setminus \{h\}} \hat{w}_i \mid A(\tau'')]$ is referring to the fact that τ'' is the $k - |I'| + 1$'st highest priority where $I' = I \setminus \{h\}$.

Case 2: For all $h \in I$, $w_h < \tau''$. Let q be the minimum priority among items in I. If $q < \tau''$ then $\hat{w}_j = 0$ for some $j \in I$ and the entire product is 0. Thus, in this case, there is no contribution to the expectation. Thus we will consider the case when $q \geq \tau''$. The probability for this event is $\prod_{i \in I} \frac{w_i}{\tau''}$. But in this case all $i \in I$ will be in S and moreover $\hat{w}_i = \tau''$ for each i. Thus the expected value of $\prod_{i \in I} \hat{w}_i = \prod_{i \in I} w_i$ as desired. □

Combining Lemma 2 and 3 the variance of the estimator $\sum_{i \in I \cap S} \hat{w}_i$ is

$$\mathbf{Var}[\sum_{i \in I \cap S} \hat{w}_i = \sum_{i \in I \cap S} \mathbf{Var}[\hat{w}_i] = \sum_{i \in I \cap S} \mathbf{E}[\hat{v}_i].$$

The advantage of this is that the variance of the estimator can be computed by examining τ and the weights of the elements in the $S \cap I$.

2 ℓ_0 Sampling

We have seen ℓ_2 sampling in the streaming setting. The ideas generalize to ℓ_p sampling to ℓ_p sampling for $p \in (0, 2)$ — see [] for instance. However, ℓ_0 sampling requires slightly different ideas. ℓ_0 sampling means that we are sampling near-uniformly from the distinct elements in the stream. Surprisingly we can do this even in the turnstile setting.

Recall that one of the applications we saw for the Count-Sketch is ℓ_2-sparse recovery. In particular we can obtain a $(1+\epsilon)$-approximation for $\mathrm{err}_2^k(\mathbf{x})$ with high-probability using $O(k \log n/\epsilon)$ words. Suppose $\mathbf{x}$ is k-sparse then $\mathrm{err}_2^k(\mathbf{x}) = 0$! It means that we can detect if $\mathbf{x}$ is k-sparse, and in fact identify the non-zero coordinates of $\mathbf{x}$, with high-probability. In fact one can prove the following stronger version.

Lemma 5 *For $1 \le k \le n$ there and $k' = O(k)$ there is a sketch $L : \mathbb{R}^n \to \mathbb{R}^{k'}$ (generated from $O(k \log n)$ random bits) and a recovery procedure that on input $L(\mathbf{x})$ has the following feature: (i) if $\mathbf{x}$ is k-sparse then it outputs $\mathbf{x}' = \mathbf{x}$ with probability 1 and (ii) if $\mathbf{x}$ is not k-sparse the algorithm detects this with high-probability.*

We will use the above for ℓ_0 sampling as follows. We will first describe a high-level algorithm that is not streaming friendly and will indicate later how it can be made stream implementable.

1. For $h = 1, \ldots, \lfloor \log n \rfloor$ let I_h be a random subsets of $[n]$ where I_j has cardinality 2^j. Let $I_0 = [n]$.
2. Let $k = \lceil 4 \log(1/\delta) \rceil$. For $h = 0, \ldots, \lfloor \log n \rfloor$, run k-sparse-recovery on $\mathbf{x}$ restricted to coordinates of I_h.
3. If any of the sparse-recoveries succeeds then output a random coordinate from the first sparse-recovery that succeeds.
4. Algorithm fails if none of the sparse-recoveries output a valid vector.

Let J be the index set of non-zero coordinates of $\mathbf{x}$. We now show that the algorithm with probability $(1 - \delta)$ succeeds in outputting a uniform sample from J. Suppose $|J| \le k$. Then $\mathbf{x}$ is recovered exactly for $h = 0$ and the algorithm outputs a uniform sample from J. Suppose $|J| > k$. We observe that $\mathbf{E}[|I_h \cap J|] = 2^h|J|/n$ and hence there is a h^* such that $\mathbf{E}[|I_{h^*} \cap J|] = 2^{h^*}|J|/n$ is between $k/3$ and $2k/3$. By Chernoff-bounds one can show that with probability at least $(1 - \delta)$, $1 \le |I_{h^*} \cap J| \le k$. For this h^* the sparse recovery will succeed and output a random coordinate of J. The formal claims are the following:

- With probability at least $(1 - \delta)$ the algorithm outputs a coordinate $i \in [n]$.
- If the algorithm outputs a coordinate i then the probability that it is not a uniform random sample is because the sparse recovery algorithm failed for some h; we can make this probability be less than $1/n^c$ for any desired constant c.

Thus, in fact we get zero-error ℓ_0 sample.

The algorithm, as described, requires one to sample and store I_h for $h = 0, \ldots, \lfloor \log n \rfloor$. In order to avoid this we can use Nisan's pseudo-random generator for small-space computation. We skip the details of this; see [2]. The overall space requirements for the above procedure can be shown to be $O(\log^2 n \log(1/\delta)$ with an error probability bounded by $\delta + O(1/n^c)$. This is near-optimal for constant δ as shown in [2].

Bibliographic Notes: The material on priority sampling is directly from [1] which describes applications, relationship to prior sampling techniques and also has an experimental evaluation. Priority sampling has shown to be "optimal" in a strong sense; see [3].

The ℓ_0 sampling algorithm we described is from the paper by Jowhari, Saglam and Tardos [2]. See a simpler algorithm in the chapter on signals by McGregor-Muthu draft book.

References

[1] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32, 2007.

[2] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. *CoRR*, abs/1012.4889, 2010.

[3] Mario Szegedy. The dlt priority sampling is essentially optimal. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 150–158. ACM, 2006.

CS 598CSC: Algorithms for Big Data	**Lecture date: Sept 25, 2014**
Instructor: Chandra Chekuri	**Scribe: Chandra Chekuri**

Suppose we have a stream $a_1, a_2, \ldots, a_n$ of objects from an ordered universe. For simplicity we will assume that they are real numbers and more over that they are distinct (for simplicity). We would like to find the k'th ranked element for some $1 \leq k \leq n$. In particular we may be interested in the median element. We will discuss exact and approximate versions of these problems. Another terminology for finding rank k elements is quantiles. Given a number ϕ where $0 < \phi \leq 1$ we would like to return an element of rank ϕn. This normalization allows us to talk about ϵ-approximate quantiles for $\epsilon \in [0, 1)$. An ϵ-approximate quantile is an element whose rank is at least $(\phi - \epsilon)n$ and at most $(\phi + \epsilon)n$. In other words we are allowing an additive error of ϵn. There is a large amount of literature on quantiles and quantile queries. In fact one of the earliest "streaming" papers is the one by Munro and Paterson [5] who described a p-pass algorithm for selection using $\tilde{O}(n^{1/p})$ space for any $p \geq 2$. They also considered the "random-order" streaming setting which has become quite popular in recent years.

The material for these lectures is taken mainly from the excellent chapter/survey by Greenwald and Khanna [1]. We mainly refer to that chapter and describe here the outline of what we covered in lectures. We will omit proofs or give sketchy arguments and refer the reader to [1].

1 Approximate Quantiles and Summaries

Suppose we want to be able to answer ϵ-approximate quantile queries over an ordered set S of n elements. It is easy to see that we can simply pick elements of rank $i|S|/k$ for $1 \leq i \leq k \simeq 1/\epsilon$ and store them as a summary and use the summary to answer any quantile query with an ϵn additive error. However, to obtain these elements we first need to do selection which we can do in the offline setting if all we want is a concise summary. The question we will address is to find a summary in the streaming setting. In the sequel we will count space usage in terms of "words". Thus, the space usage for the preceding offline summary is $\Theta(1/\epsilon)$.

We will see two algorithms. The first will create an ϵ-approximate summary [4] with space $O(\frac{1}{\epsilon}\log^2(\epsilon n))$ and is inspired by the ideas from the work of Munro and Paterson. Greenwald and Khanna [2] gave a different summary that uses $O(\frac{1}{\epsilon}\log(\epsilon n))$ space.

Following [1] we will think of a quantile summary Q as storing a set of elements $\{q_1, q_2, \ldots, q_\ell\}$ from S along with an interval $[\mathbf{rmin}_Q(q_i), \mathbf{rmax}_Q(q_i)]$ for each q_i; $\mathbf{rmin}_Q(q_i)$ is a lower bound on the rank of q_i in S and $\mathbf{rmax}_Q(q_i)$ is an upper bound on the rank of q_i in S. It is convenient to assume that $q_1 < q_2 < \ldots < q_\ell$ and moreover that q_1 is the minimum element in S and that q_ℓ is the maximum element in S. For ease of notation we will simply use Q to refer to the quantile summary and also the (ordered) set $\{q_1, q_2, \ldots, q_\ell\}$.

Our first question is to ask whether a quantile summary Q can be used to give ϵ-approximate quantile queries. The following is intuitive and is worthwhile proving for oneself.

Lemma 1 *Suppose Q is a quantile summary for S such that for $1 \leq i < \ell$, $\mathbf{rmax}_Q(q_{i+1}) - \mathbf{rmin}_Q(q_i) \leq 2\epsilon|S|$. Then Q can be used for ϵ-approximate quantile queries over S.*

In the following, when we say that Q is an ϵ-approximate quantile summary we will implicitly be using the condition in the preceding lemma.

Given a quantile summary Q' for a multiset S' and a quantile summary Q'' for a multiset S'' we would like to combine Q' and Q'' into a quantile summary Q for $S = S' \cup S''$; by union here we view S as a multiset. Of course we would like to keep the approximation of the resulting summary similar to those of Q' and Q''. Here a lemma which shows that indeed we can easily combine.

Lemma 2 *Let Q' be an ϵ'-approximate quantile summary for multiset S' and Q'' be an ϵ''-approximate quantile summary for multiset S''. Then $Q = Q' \cup Q''$ yields an ϵ-approximate quantile summary for $S = S' \cup S''$ where $\epsilon \leq \frac{\epsilon' n' + \epsilon'' n''}{n' + n''} \leq \max\{\epsilon', \epsilon''\}$ where $n' = |S'|$ and $n'' = |S''|$.*

We will not prove the correctness but describe how the intervals are constructed for Q from those for Q' and Q''. Suppose $Q' = \{x_1, x_2, \ldots, x_a\}$ and $Q'' = \{y_1, y_2, \ldots, y_b\}$. Let $Q = Q' \cup Q'' = \{z_1, z_2, \ldots, z_{a+b}\}$. Consider some $z_i \in Q$ and suppose $z_i = x_r$ for some $1 \leq r \leq a$. Let y_s be the largest element of Q'' smaller than x_r and y_t be the smallest element of Q'' larger than x_r. We will ignore the cases where one or both of y_s, y_t are not defined. We set

$$\mathbf{rmin}_Q(z_i) = \mathbf{rmin}_{Q'}(x_r) + \mathbf{rmin}_{Q''}(y_s)$$

and

$$\mathbf{rmax}_Q(z_i) = \mathbf{rmax}_{Q'}(x_r) + \mathbf{rmax}_{Q''}(y_t) - 1.$$

It is easy to justify the above as valid intervals. One can then prove that with these settings, for $1 \leq i < a + b$ the following holds:

$$\mathbf{rmax}_Q(z_{i+1}) - \mathbf{rmin}_Q(z_i) \leq 2\epsilon(n' + n'').$$

We will refer to the above operation as COMBINE(Q', Q''). The following is easy to see.

Lemma 3 *Let $Q_1, \ldots, Q_h$ be ϵ-approximate quantile summaries for $S_1, S_2, \ldots, S_h$ respectively. Then $Q_1, Q_2, \ldots, Q_h$ can be combined in any arbitrary order to obtain an ϵ-approximate quantile summary $Q = Q_1 \cup \ldots \cup Q_h$ for $S_1 \cup \ldots \cup S_h$.*

Next we discuss the PRUNE(Q, k) operation on a quantile summary Q that reduces the size of Q to $k + 1$ while losing a bit in the approximation quality.

Lemma 4 *Let Q' be an ϵ-approximate quantile summary for S. Given an integer parameter k there is a quantile summary $Q \subseteq Q'$ for S such that $|Q| \leq k + 1$ and it is $(\epsilon + \frac{1}{2k})$-approximate.*

We sketch the proof. We simply query Q' for ranks $1, |S|/k, 2|S|/k, \ldots, |S|$ and choose these elements to be in Q. We retain their **rmin** and **rmax** values from Q'.

$$\mathbf{rmax}_Q(q_{i+1}) - \mathbf{rmin}_Q(q_i) \leq i|S|/k + \epsilon|S| - ((i-1)|S|/k - \epsilon|S|) \leq |S|/k + 2\epsilon|S|.$$

1.1 An $O(\frac{1}{\epsilon}\log^2(\epsilon n))$ space algorithm

The idea is inspired by the Munro-Paterson algorithm and was abstracted in the paper by Manku et al. We will describe the idea in an offline fashion though it can be implemented in the streaming setting. We will use several quantile summaries with k elements each for some parameter k, say ℓ of them. Each summary of size k will be called a buffer. We will need to reuse these buffers as more elements in the stream arrive; buffers will combined and pruned, in other words "collapsed" into a single buffer of the same size k. Pruning introduces error.

Assume n/k is a power of 2 for simplicity. Consider a complete binary tree with n/k leaves where each leaf corresponds to k consecutive elements of the stream. Think of assigning a buffer of size k to obtain a 0-error quantile summary for those k elements; technically we need $k+1$ elements but we will ignore this minor additive issue for sake of clarity of exposition. Now each internal node of the tree corresponds to a subset of the elements of the stream. Imagine assigning a buffer of size k to each internal node to maintain an approximate quantile summary for the elements of the stream in the sub-tree. To obtain a summary at node v we combine the summaries of its two children v_1 and v_2 and prune it back to size k introducing and additional $1/(2k)$ error in the approximation. The quantile summary at the root of size k will be our final summary that we output for the stream.

Our first observation is that in fact we can implement the tree-based scheme with $\ell = O(h)$ buffers where h is the height of the tree. Note that $h \simeq \log(n/k)$. The reason we only need $O(h)$ buffers is that if need a new buffer for the next k elements in the stream we can collapse two buffers corresponding to the children of an internal node — hence, at any time we need to maintain only one buffer per level of the tree (plus a temporary buffer to do the collapse operation).

Consider the quantile summary at the leaves. They have error 0 since we store all the elements in the buffer. However at each level the error increases by $1/(2k)$. Hence the error of the summary at the root is $h/(2k)$. Thus, to obtain an ϵ-approximate quantile summary we need $h/(2k) \leq \epsilon$. And $h = \log(n/k)$. One can see that for this to work out it suffices to choose $k > \frac{1}{2\epsilon}\log(2\epsilon n)$.

The total space usage is $\Theta(hk)$ and $h = \log(n/k)$ and thus the space usage is $O(\frac{1}{\epsilon}\log^2(\epsilon n))$.

One can choose d-ary trees instead of binary trees and some optimization can be done to improve the constants but the asymptotic dependence on ϵ does not improve with this high-level scheme.

1.2 An $O(\frac{1}{\epsilon}\log(\epsilon n)$ space algorithm

We now briefly describe the Greenwald-Khanna algorithm that obtains an improved space bound. The GK algorithm maintains a quantile summary as a collection of s tuples $t_0, t_2, \ldots, t_{s-1}$ where each tuple t_i is a triple (v_i, g_i, Δ_i): (i) a value v_i that is an element of the ordered set S (ii) the value g_i which is equal to $\mathbf{rmin}_{\mathrm{GK}}(v_i) - \mathbf{rmin}_{\mathrm{GK}}(v_{i-1})$ (for $i = 0$, $g_i = 0$) and (iii) the value Δ_i which equals $\mathbf{rmax}_{\mathrm{GK}}(v_i) - \mathbf{rmin}_{\mathrm{GK}}(v_i)$. The elements $v_0, v_1, \ldots, v_{s-1}$ are in ascending order and moreover v_0 will the minimum element in S and v_{s-1} is the maximum element in S. Note that $n = \sum_{j=1}^{s-1} g_j$. The summary also stores n the number of elements seen so far. With this set up we note that $\mathbf{rmin}_{\mathrm{GK}}(v_i) = \sum_{j \leq i} g_j$ and $\mathbf{rmax}_{\mathrm{GK}}(v_i) = \Delta_i + \sum_{j \leq i} g_j$.

Lemma 5 *Suppose Q is a GK quantile summary for a set $|S|$ such that $\max_i(g_i + \Delta_i) \leq 2\epsilon|S|$ then it can be used to answer quantile queries with ϵn additive error.*

The query can be answered as follows. Given rank r, find i such that $r - \mathbf{rmin}_{\mathrm{GK}}(v_i) \leq \epsilon n$ and $\mathbf{rmax}_{\mathrm{GK}}(v_i) - r \leq \epsilon n$ and output v_i; here n is the current size of S.

The quantile summary is updated via two operations. When a new element v arrives it is inserted into the summary. The quantile summary is compressed by merging consecutive elements to keep the summary within the desired space bounds.

We now describe the INSERT operation that takes a quantile summary Q and inserts a new element v. First we search over the elements in Q to find an i such that $v_i < v < v_{i+1}$; the case when v is the new smallest element or the new largest element are handled easily. A new tuple $t = (v, 1, \Delta)$ with $\Delta = \lfloor 2\epsilon n \rfloor - 1$ is added to the summary where t becomes the new $(i+1)$st tuple. Note that here n is the current size of the stream. It is not hard to see if the summary Q before arrival of v satisfied the condition in Lemma 5 then Q satisfies the condition after inserting the tuple (note that n increased by 1). We note that that the first $1/(2\epsilon)$ elements are inserted into the summary with $\Delta_i = 0$.

Compression is the main ingredient. To understand the operation it is helpful to define the notion of a *capacity* of a tuple. Note that when v arrived $t = (v, 1, \Delta)$ is inserted where $\Delta \simeq 2\epsilon n'$ where n' is the time when v arrived. At time $n > n'$ the capacity of the tuple t_i is defined as $2\epsilon n - \Delta_i$. As n grows, the capacity of the tuple increases since we are allowed to have more error. We can merge tuples $t_{i'}, t_{i'+1}, \ldots, t_i$ into t_{i+1} at time n (which means we eliminate $t_{i'}, \ldots, t_i$) while ensuring the desired precision if $\sum_{j=i'}^{i+1} g_j + \Delta_{i+1} \leq 2\epsilon n$; g_{i+1} is updated to $\sum_{j=i'}^{i+1} g_j$ and Δ_{i+1} does not change. Note that this means that Δ of a tuple does not change once it is inserted.

Note that the insertion and merging operations preserve correctness of the summary. In order to obtain the desired space bound the merging/compression has to be done rather carefully. We will not go into details but mention that one of the key ideas is to keep track of the capacity of the tuples in geometrically increasing intervals and to ensure that the summary retains only a small number of tuples per interval.

2 Exact Selection

We will now describe a p-pass deterministic algorithm to select the rank k element in a stream using $\tilde{O}(n^{1/p})$-space; here $p \geq 2$. It is not hard to show that for $p = 1$ any deterministic algorithm needs $\Omega(n)$ space; one has to be a bit careful in arguing about bits vs words and the precise model but a near-linear lower bound is easy. Munro and Paterson described the $\tilde{O}(n^{1/p})$-space algorithm using p passes. We will not describe their precise algorithm but instead use the approximate quantile based analysis.

We will show that given space s and a stream of n items the problem can be effectively reduced in one pass to selecting from $O(n \log^2 n/s)$ items.

Suppose we can do the above. Choose $s = n^{1/p}(\log n)^{2-2/p}$. After i passes the problem is reduced to $n^{\frac{p-i}{p}}(\log n)^{\frac{2i}{p}}$ elements. Setting $i = p - 1$ we see that the number of elements left for the p'th pass is $O(s)$. Thus all of them can be stored and selection can be done offline.

We now describe how to use one pass to reduce the effective size of the elements under consideration to $O(n \log^2 n/s)$. The idea is that we will be able to select two elements a_1, b_1 from the stream such that $a_1 < b_1$ and the k'th ranked element is guaranteed to be in the interval $[a_1, b_1]$. Moreover, we are also guaranteed that the number of elements between a and b in the stream is $O(n \log^2 n/s)$. a_1 and b_1 are the left and right filter after pass 1. Initially $a_0 = -\infty$ and $b_0 = \infty$. After i passes we will have filters a_i, b_i. Note that during the $(i + 1)$st pass we can compute the exact rank of a_i and b_i.

How do we find a_1, b_1? We saw how to obtain an ϵ-approximate summary using $O(\frac{1}{\epsilon} \log^2 n)$ space. Thus, if we have space s, we can set $\epsilon' = \log^2 n/s$. Let $Q = \{q_1, q_2, \ldots, q_\ell\}$ be ϵ'-approximate quantile summary for the stream. We query Q for $r_1 = k - \epsilon' n - 1$ and $r_2 = k + \epsilon' n + 1$ and obtain a_1 and b_1 as the answers to the query (here we are ignoring the corner cases where $r_1 < 0$ or $r_2 > n$). Then, by the ϵ'-approximate guarantee of Q we have that the rank k element lies in the interval $[a_1, b_1]$ and moreover there are at most $O(\epsilon' n)$ elements in this range.

It is useful to work out the algebra for $p = 2$ which shows that the median can be computed in $O(\sqrt{n} \log^2 n)$ space.

2.1 Random Order Streams

Munro and Paterson also consider the random order stream model in their paper. Here we assume that the stream is a random permutation of an ordered set. It is also convenient to use a different model where the the i'th element is a real number drawn independently from the interval $[0, 1]$. We can ask whether the randomness can be taken advantage of. Indeed one can. They showed that with $O(\sqrt{n})$ space one can find the median with high probability. More generally they showed that in p passes one can find the median with space $O(n^{1/(2p)})$. Even though this space bound is better than for adversarial streams it still requires $\Omega(\log n)$ passes is we have only poly-logarithmic space, same as the adversarial setting. Guha and McGregor [3] showed that in fact $O(\log \log n)$ passes suffice (with high probability).

Here we describe the Munro-Paterson algorithm; see also `http://polylogblog.wordpress.com/2009/08/30/bite-sized-streams-exact-median-of-a-random-order-stream/`.

The algorithm maintains a set S of s consecutively ranked elements in the stream seen so far. It maintains two counters ℓ for the number of elements less then $\min S$ (the min element in S) which have been seen so far and h for the number of elements larger than $\max S$ which have been so far. It tries to maintain $h - \ell$ as close to 0 as possible to "capture" the median.

```
MunroPaterson($s$):

$n \leftarrow 0$
$S \leftarrow \emptyset$
$\ell, h \leftarrow 0$
While (stream is not empty) do
  $n \leftarrow n + 1$
  $a$ is the new element
  if ($a < \min S$) then $\ell \leftarrow \ell + 1$
  else if ($a > \max S$) then $h \leftarrow h + 1$
  else add $a$ to $S$
  if ($|S| = s + 1$)
      if ($h < \ell$) discard $\max S$ from $S$ and $h \leftarrow h + 1$
      else discard $\min S$ from $S$ and $\ell \leftarrow \ell + 1$
endWhile
if $1 \leq (n + 1)/2 - \ell \leq s$ then
  return $(n + 1)/2 - \ell$-th smallest element in $S$ as median
else return FAIL.
```

To analyze the algorithm we consider the random variable $d = h - \ell$ which starts at 0. In the first s iterations we simply fill up S to capacity and $h - \ell$ remains 0. After that, in each step d is either incremented or decremented by 1. Consider the start of iteration i when $i > s$. The total number of elements seen prior to i is $i - 1 = \ell + h + s$. In iteration i, since the permutation is random, the probability that a_i will be larger than $\max S$ is precisely $(h + 1)/(h + s + 1 + \ell)$. The probability that a_i will be smaller than $\min S$ is precisely $(\ell + 1)/(h + s + 1 + \ell)$ and thus with probability $(s - 1)/(h + s + 1 + \ell)$, a_i will be added to S.

Note that the algorithm fails only if $|d|$ at the end of the stream is greater than s. A sufficient condition for success is that $|d| \leq s$ *throughout* the algorithm. Let $p_{d,i}$ be the probability that $|d|$ increases by 1 conditioned on the fact that $0 < |d| < s$. Then we see that $p_{d,i} \leq 1/2$. Thus the process can be seen to be similar to a random walk on the line and some analysis shows that if we choose $s = \Omega(\sqrt{n})$ then with high probability $|d| < s$ throughout. Thus, $\Omega(\sqrt{n})$ space suffices to find the median with high probability when the stream is in random order.

Connection to CountMin sketch and deletions: Note that when we were discussion frequency moments we assume that the elements were drawn from a $[n]$ where n was known in advance while here we did not assume anything about the elements other than the fact that they came from an ordered universe (apologies for the confusion in notation since we used m previously for length of stream). If we know the range of the elements in advance and it is small compared to the length of the stream then CountMin and related techniques are better suited and provide the ability to handle deletions. The GK summary can also handle some deletions. We refer the reader to [1] for more details.

Lower Bounds: For median selection Munro and Paterson showed a lower bound of $\Omega(n^{1/p})$ on the space for p passes in a restricted model of computation. Guha and McGregor showed a lower bound of $\Omega(n^{1/p}/p^6)$ bits without any restriction. For random order streams $O(\log \log n)$ passes suffice with polylog(n) space for exact selection with high probability [3]. Moreover $\Omega(\log \log n)$ passes are indeed necessary; see [3] for references and discussion.

Bibliographic Notes: See the references for more information.

References

[1] Michael Greenwald and Sanjeev Khanna. Quantiles and equidepth histograms over streams. Available at `http://www.cis.upenn.edu/~mbgreen/papers/chapter.pdf`. To appear as a chapter in a forthcoming book on Data Stream Management.

[2] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD Record*, volume 30, pages 58–66. ACM, 2001.

[3] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

[4] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *ACM SIGMOD Record*, volume 27, pages 426–435. ACM, 1998.

[5] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.