CPSC 536N: Randomized Algorithms	2014-15 Term 2
Lecture 15	
Prof. Nick Harvey	*University of British Columbia*

1 Streaming Algorithms for the Distinct Elements Problem

In this lecture we consider another problem in the streaming model. Recall that the input is a sequence $(a_1, a_2, \ldots, a_m)$ of indices with each $a_i \in [n]$. The frequency vector is $f \in \mathbb{Z}^n$, where

$$f_j \;=\; |\{\; i \;:\; a_i = j \;\}| \;=\; \text{number of occurrences of } j \text{ in the stream.}$$

Define $D = \{\; a_i \;:\; i = 1, \ldots, m \;\}$ be the distinct elements that occur in the stream. Let $d = |D|$. The goal of today is to estimate

$$d \;=\; |\{\; j \;:\; f_j > 0 \;\}| \;=\; \text{number of distinct elements in the stream.}$$

This problem has a huge number of applications in networking, databases, computer systems, etc.

The number of distinct elements is sometimes called the "ℓ_0-norm" of the vector f because it happens to equal $\lim_{p \searrow 0} \|f\|_p^p$. It is also sometimes denoted $\|f\|_0$. However $\|\cdot\|_0$ is not an actual norm (it doesn't satisfy $\|\alpha f\|_0 = \alpha \|f\|_0$ for $\alpha \geq 0$).

1.1 Idea #1: Random sampling

A natural idea for this problem is to do some sort of random sampling. Suppose we chose some small random set $U \subseteq [n]$, and keep track of the number of distinct elements in the stream that lie in U. This might allow us to estimate d.

For example, that every element in $[n]$ is added to U with probability p. Then the expected number of elements in $D \cap U$ is dp. If we have $p = 1/d$ then this expectation is 1. So if $|D \cap U| > 1$ then we could regard that as evidence that $|D| \geq 1/p$.

This idea can be made to work, but the details get a bit messy because the "right" probability p used to generate U requires knowing d.

References: Indyk Lecture 3, Madry Lecture 16.

1.2 Idea #2: Making the distribution uniform

Let's consider a different approach. Consider the set D as a subset of $[n]$. The elements of D need not be nicely spread out — they may be bunched together, or make some other sort of unpleasant pattern inside $[n]$. But suppose we hash the elements of $[n]$ to the interval $[0, 1]$ (which is essentially the same as hashing to long binary strings). Then the elements of D get hashed to d independent and uniformly distributed values in $[0, 1]$.

That seems like a nice distribution. Can we compute some of its properties in low space, and use those to estimate d? One idea is that the *minimum* of the hash values should roughly equal $1/d$.

This suggests the following algorithm. It keeps track of X, the minimum hash value seen so far. At the end of the stream, we should have $X \approx 1/d$, so $1/X$ should be a reasonable estimate for d.

The only space required is to store X (and the hash function). We just need X to have enough precision so that the algorithm's error is due to the variance of X, not due to the precision with which X is stored. Presumably storing X with $O(\log n)$ bits should be enough.

1.3 Rough analysis via order statistics

To get a rough idea of how well this should work, let's hold our noses and dig in to the statistics literature. The t^{th} smallest value of a random sample is called the t^{th} order statistic. As one might expect, these have been intensively studied.

Our variable X is distributed as the first order statistic of d independent samples from the uniform distribution on $[0, 1]$. According to Wikipedia, X has the Beta distribution $B(1, d)$. The mean of this distribution equals $1/(d+1)$. But in order for $1/X$ to give a good estimate for d, we need to X to have small variance.

The variance of the distribution $B(1, d)$ is also known:

$$\mathrm{Var}\left[\, X \,\right] \;=\; \frac{d}{(d+1)^2(d+2)} \;\approx\; 1/d^2,$$

so the standard deviation is $\sigma = \sqrt{\mathrm{Var}\left[\, X \,\right]} \approx 1/d$. So we would expect X to frequently take values throughout the interval $\mathrm{E}\left[\, X \,\right] + [-\sigma, \sigma] \approx [0, 2/d]$. This does not look promising!

1.4 An improved idea

Why did the previous idea not work well? Intuitively, random variables that are influenced by *many* independent random variables should be well-concentrated. The first order statistic X does not have that property: perturbing just one of the random values (the smallest one) is enough to significantly affect the value of X. We need a well-concentrated statistic, which should depend strongly on many of the random values.

A natural idea is to consider the t^{th} order statistic for some $t > 1$, which would seem to depend rather strongly on the t smallest values. So now let X be the t^{th} smallest value in the random sample. According to Wikipedia, X has the Beta distribution $B(t, d+1-t)$ and

$$\begin{aligned}
\mathrm{E}\left[\, X \,\right] \;&=\; \frac{t}{d+1} \\
\mathrm{Var}\left[\, X \,\right] \;&=\; \frac{t(d+1-t)}{(d+1)^2(d+2)}
\end{aligned}$$

So, by Chebyshev's inequality, we should have

$$\Pr\left[\, |X - \mathrm{E}\left[\, X \,\right]| \geq \epsilon\, \mathrm{E}\left[\, X \,\right] \,\right] \;\leq\; \frac{\mathrm{Var}\left[\, X \,\right]}{(\epsilon\, \mathrm{E}\left[\, X \,\right])^2} \;=\; \frac{(d+1-t)}{\epsilon^2 t(d+2)} \;\leq\; \frac{1}{\epsilon^2 t}.$$

From an accuracy standpoint this seems promising, so long as $t \gg 1/\epsilon^2$. From an algorithmic standpoint, this seems promising too: keeping track of the t smallest values in a random sample only requires $O(t)$ words of space.

2 The actual algorithm

The main flaw of the previous discussion is that it assumes that the hash values are *mutually independent*. In order to provably implement the algorithm in low space, we will need to use a low-independence hash function. So we will need to do a similar analysis from scratch, without making use of the Beta distribution (which only arises in the mutually independent case).

Our previous discussion imagined hashing to real numbers in $[0, 1]$, but it is more convenient to hash to integers, so let us scale everything up by a factor of N and hash to the set $[N] = \{1, \ldots, N\}$. We still want moderately high precision, so we will need N to be somewhat large. Formally, let us pick a pairwise independent hash function $h = h_s : [n] \to [N]$ where N is a power of two in $[n, 2n]$. Our construction in Lecture 13 will allow us to do this with a random seed that is a uniformly random bitstring of length $O(\log n)$.

Algorithm 1: The streaming algorithm for estimating d, the number of distinct elements. It achieves $(1 + \epsilon)$-multiplicative error with probability at least $1 - \delta$.

Let N be the next power of two exceeding n
Let $t \leftarrow 12/(\delta\epsilon^2)$
Pick a pairwise independent hash function $h : [n] \to [N]$
$\triangleright$ *The set S always contains the t smallest hash values seen so far in the stream*
Initialize $S \leftarrow \emptyset$
for $i = 1, \ldots, m$ **do**
    Receive the element $j = a_i \in [n]$
    Add $h(j)$ to S if necessary
if $|S| < t$ **then**
    Output $|S|$
else
    Let X be the largest hash value in S, which is the t^{th} smallest in the entire stream
    Output tN/X

Let us first state explicitly a trivial fact concerning variance of indicator random variables.

Fact 1 *Let Y be an indicator (Bernoulli) random variable. Then* $\mathrm{Var}[Y] \le \Pr[Y = 1]$.

PROOF: $\mathrm{Var}[Y] = \mathrm{E}[Y^2] - \mathrm{E}[Y]^2 \le \mathrm{E}[Y^2] = \mathrm{E}[Y]$. $\square$

Claim 2 $\Pr[tN/X > (1 + \epsilon)d] \le 4/(\epsilon^2 t)$.

PROOF: The event is equivalent to $X < tN/\big((1 + \epsilon)d\big)$. This event occurs if and only if, for at least t elements $j \in D$, we have

$$h(j) \;<\; \frac{tN}{(1+\epsilon)d} \;\le\; \frac{(1-\epsilon/2)tN}{d}.$$

Define Z_j to be the indicator of the event

$$\left\{ h(j) \le \frac{(1-\epsilon/2)tN}{d} \right\}.$$

Then

$$\begin{aligned} \mathrm{E}[Z_j] &\le (1-\epsilon/2)t/d \qquad \text{(since } h(j) \text{ is uniform on } [N]) \\ \mathrm{Var}[Z_j] &\le (1-\epsilon/2)t/d \ < \ t/d \qquad \text{(by Fact 1).} \end{aligned}$$

Define $Z = \sum_{j \in D} Z_j$. Linearity of expectation and pairwise independence of the Z_j implies

$$\mathrm{E}[Z] \le (1-\epsilon/2)t \tag{1}$$

$$\mathrm{Var}[Z] = \sum_{j \in D} \mathrm{Var}[Z_j] \le t. \tag{2}$$

Now we use Chebyshev's inequality to obtain

$$\begin{aligned} \Pr[tN/X > (1+\epsilon)d] &\le \Pr[Z \ge t] \\ &\le \Pr[|Z - \mathrm{E}[Z]| \ge \epsilon t/2] \qquad \text{(by (1))} \\ &\le \frac{\mathrm{Var}[Z]}{(\epsilon t/2)^2} \qquad \text{(Chebyshev's inequality)} \\ &\le \frac{4}{\epsilon^2 t} \qquad \text{(by (2)).} \end{aligned}$$

□

Claim 3 *Assume that* $N \ge 2n/(t\epsilon)$ *and* $\epsilon \le 1/2$. *Then* $\Pr[tN/X < (1-\epsilon)d] \le 8/(\epsilon^2 t)$.

Proof: The event is equivalent to $X > tN/\big((1-\epsilon)d\big)$. This event occurs if and only if, for fewer than t elements $j \in D$, we have $h(j) \le \frac{tN}{(1-\epsilon)d}$. Define Z_j to be the indicator of the event

$$\left\{ h(j) \le \frac{tN}{(1-\epsilon)d} \right\}.$$

This time will will need both upper and lower bounds on $\mathrm{E}[Z_j]$. Due to rounding issues, we lose a little bit in the lower bound. Since $N \ge d/(t\epsilon)$, we obtain

Upper bound: $\mathrm{E}[Z_j] \le \dfrac{t}{(1-\epsilon)d} \le \dfrac{2t}{d}$ (since $\epsilon \le 1/2$)

Lower bound: $\mathrm{E}[Z_j] \ge \dfrac{t}{(1-\epsilon)d} - \dfrac{1}{N} \ge \dfrac{(1+\epsilon)t}{d} - \dfrac{1}{N} \ge \dfrac{(1+\epsilon/2)t}{d}$ (since $N \ge 2n/(t\epsilon)$).

By Fact 1, $\mathrm{Var}[Z_j] \le 2t/d$. Define $Z = \sum_{j \in D} Z_j$. Linearity of expectation and pairwise independence of the Z_j imply

$$\mathrm{E}[Z] \ge (1+\epsilon/2)t \tag{3}$$

$$\mathrm{Var}[Z] = \sum_{j \in D} \mathrm{Var}[Z_j] \le 2t. \tag{4}$$

Now we use Chebyshev's inequality to obtain

$$\begin{aligned} \Pr[tN/X < (1-\epsilon)d] &\le \Pr[Z < t] \\ &\le \Pr[|Z - \mathrm{E}[Z]| \ge \epsilon t/2] \qquad \text{(by (3))} \\ &\le \frac{\mathrm{Var}[Z]}{(\epsilon t/2)^2} \qquad \text{(Chebyshev's inequality)} \\ &\le \frac{8}{\epsilon^2 t} \qquad \text{(by (4)).} \end{aligned}$$

□

Combining Claim 2 and Claim 3, we see that

$$\Pr[\text{algorithm's output} \notin [1-\epsilon, 1+\epsilon]\cdot d] \;\leq\; \frac{4}{\epsilon^2 t} + \frac{8}{\epsilon^2 t}.$$

Thus, if we choose $t = 12/(\delta\epsilon^2)$, we obtain that

$$\Pr[\text{algorithm's output} \in [1-\epsilon, 1+\epsilon]\cdot d] \;\geq\; 1-\delta.$$

A small detail. There is one detail that was swept under the rug. If $|S| < t$, the algorithm just outputs $|S|$. We should also check the quality of that estimate. As an exercise, one should show that if $|S| < t$ then, with high probability, all distinct elements in the stream have different hash values.

2.1 Space Analysis.

Let us now consider how much space the algorithm needs in order to achieve these guarantees.

The set S. There are t coordinates, each of which uses $O(\log n)$ bits.

The hash function h. To represent the hash function h we only need to store the random seed s. In our hash construction, each seed uses $O(\log N) = O(\log n)$ bits of space.

Thus, the total space is $O(t \log n) = O(\log(n)/\delta\epsilon^2)$ bits.

3 The State of the Art

The method that we presented is from a paper of Bar-Yossef, Jayram, Kumar, Sivakumar and Trevisan. In practice, distinct element estimation is widely used. A method called HyperLogLog uses roughly $O(\epsilon^{-2} \log\log n)$ bits of space to achieve $(1+\epsilon)$ accuracy. A research paper from Google makes this algorithm very practical.

From a theoretical perspective, optimal guarantees are known. The lower bound is:

Theorem 4 (Alon-Mattias-Szegedy 1999, Indyk-Woodruff 2003) *Any randomized streaming algorithm that achieves $(1+\epsilon)$-multiplicative error for the distinct elements problem with constant probability requires $\Omega(\log(n) + \epsilon^{-2})$ bits of space.*

In contrast, the space usage of our algorithm has the *product* of $O(\log n)$ and $O(\epsilon^{-2})$, not the sum. It turns out that the sum is the right answer.

Theorem 5 (Kane-Nelson-Woodruff 2010) *There is a randomized streaming algorithm that achieves $(1+\epsilon)$-multiplicative error for the distinct elements problem with constant probability and uses only $O(\log(n) + \epsilon^{-2})$ bits of space.*