CSE 290A: Randomized algorithms
Lecture 14: Streaming algorithms (I)

Lecturer: C. Seshadhri
University of California, Santa Cruz
Scribe: Yatong Chen (ychen592@ucsc.edu)
May 14th, 2020

1 Streaming Algorithms

1.1 Definition and Motivation

In the streaming setting, we have a fixed universe of elements/items $[n]$. We receive a sequence of inputs, each an element in $[n]$, and must process these items in the given order and with limited memory.

More formally, our input is an array of length m containing items in $[n]$. A *streaming algorithm* processes the array using a single pass or a constant number of passes over the array. Typically we want the algorithm to use space $\ll m, n$.

The constraints of the streaming setting closely capture the real constraints faced by "big data" applications, and the algorithms we will discuss are in fact used in industry.

1.2 Models of Randomness

Before proceeding, we mention that theorists and practitioners in streaming algorithms sometimes operate under different models, in terms of what kinds of randomness are available to the algorithm:

- In the more practical model, we assume that there exist truly random hash functions. For example, certain commonly-used cryptographic hash functions have not been proven to be truly random, but in this model, we may simply assume that they are.
- In the more theoretical model, randomness is only available in the form of uniform random bits, and if the algorithm wants to use a random hash function, it must explicitly construct it.

Although the theoretical setting makes fewer assumptions and recognizes that constructing hash functions is challenging, it often requires techniques that cause the resulting streaming algorithms to be complicated and impractical. In the remainder of this lecture, we will work in the practical model; in the next lecture, we will discuss how to start from the more theoretical setting and work to remove the practical assumptions.

2 Counting Distinct Elements

The most classic problem in streaming is the task of counting the number of distinct elements in a given array. This can easily be done with m or n bits (to do it in n bits, simply build a bitmap, which is an array in which the i-th element is 1 iff the element i is in the input stream). The question is: How can we do this in $\ll n$ space?

Let d be the number of distinct elements. A simple observation is that, since $d \leq n$, we can store d using just $\log n$ bits.

Amazingly, we can actually reduce the space complexity down to $\log \log n$ bits. The idea, developed in ([Morris 77, Flajolet-Martin 85, Alon-Matias-Szegedy 99]), is that instead of directly estimating d, we try to estimate $\log d$. Algorithms that employ this strategy are referred to as "approximate counting," "probablistic counting," or simply the "LogLog algorithm" (with HyperLogLog being the most recent version).

2.1 Preliminaries

Before describing the algorithm, we assume that we have a hash function h that maps each element to a bitstring of length s, i.e.

$$h : [n] \mapsto \{0,1\}^s$$

We assume that h is truly random, which means that for each $i \in [n]$, $h(i)$ is chosen uniformly at random from $\{0,1\}^s$.

Next, for each $i \in [n]$, let $r(i)$ be the number of trailing 0's in $h(i)$. For example,

$$h(42) = 010011001\underbrace{000}_{3 \text{ zeros}}$$

so in this case, $r(42) = 3$. A useful observation about this quantity $r(i)$ is that:

$$\Pr[r(i) \geq l] = \frac{1}{2^l}$$

2.2 The PC Algorithm

We are now ready to describe the probabilistic counting (PC) algorithm. Like many streaming algorithms, the PC algorithm maintains a *sketch*, or summary, of the stream at every step. For the PC algorithm, this sketch is simply the maximum value of $r(i)$ seen thus far. The pseudocode is as follows:

Algorithm 1 PC Algorithm

```
z ← 0
for i in the stream do
    compute r(i)
    z ← max(z, r(i))
end for
return 2^z                    {as the estimate of the number of distinct elements}
```

This algorithm has a few niceties. First, repeated occurrences of the same element do not affect the value of z, since the same element i will always hash to the same value $h(i)$ and thus have the same $r(i)$. Second, the sketch z can be easily combined with others: if we have a collection of sketches $z_1, z_2, \ldots$ from different streams and wish to compute the number of distinct elements in the combined stream, we can simply take the maximum of these individual sketches.

Let's now prove the correctness of the algorithm. Recall that $\Pr[r(i) \geq l] = 2^{-l}$. Let X_l be the indicator random variable for whether exists an element i in the stream such that $r(i) \geq l$. Then we have:

$$\begin{aligned}
\Pr[X_l = 1] &= \Pr\left[\bigcup_i (r(i) \geq l)\right] \\
&= 1 - \Pr\left[\bigcap_i (r(i) < l)\right] \\
&= 1 - (1 - 2^{-l})^d \\
\Pr[X_l = 0] &= (1 - 2^{-l})^d
\end{aligned}$$

Then we can compute the probability that $z > \lceil \lg d \rceil + c$ for some $c > 0$ and use the fact that $(1-x)^d \geq 1 - dx$ (when $|dx| < 1$):

$$\Pr\left[X_{\lceil \lg d \rceil + c} = 1\right] = 1 - \left(1 - \frac{1}{2^{\lceil \lg d \rceil + c}}\right)^d \leq \frac{d}{2^{\lceil \lg d \rceil + c}} \leq 2^{-c}$$

On the other hand, we can compute the probability that $z < \lceil \lg d \rceil - c$ and use the fact that $1 - x \leq e^{-x}$:

$$\Pr\left[X_{\lceil \lg d \rceil - c} = 0\right] = \left(1 - 2^{-\lceil \lg d \rceil + c}\right)^d \leq \exp\left(-\frac{d \times 2^c}{2^{\lceil \lg d \rceil}}\right) \leq \exp\left(-2^{c-1}\right)$$

These inequalities give exponentially strong bounds on how likely we are to get an estimate that's within an additive error of c from the true count. In particular, if we take $c = 2$, then we have:

$$\begin{aligned}
\Pr\left[X_{\lceil \lg d \rceil + c} = 1\right] &= \Pr\left[z > \lceil \lg d \rceil + 2\right] \leq 2^{-2} = \frac{1}{4} \\
\Pr\left[X_{\lceil \lg d \rceil - c} = 0\right] &= \Pr\left[z < \lceil \lg d \rceil - 2\right] \leq \exp(-2) \leq \frac{1}{8}
\end{aligned}$$

By the union bound, we have:

$$\Pr\left[|z - \lceil \lg d \rceil| \leq 2\right] \geq 1 - \left(\frac{1}{4} + \frac{1}{8}\right) \geq \frac{2}{3}$$

In other words, we have:

$$\frac{1}{4} \cdot 2^{\lceil \lg d \rceil} \leq 2^z \leq 4 \cdot 2^{\lceil \lg d \rceil}$$

namely we get an 8-approximation to d with probability at least $2/3$. Then we can use the standard median trick to boost the probability of success.

2.3 Storage requirement

The storage required for the PC algorithm is simply the space needed to store z, and since we have $z \leq \lceil \lg d \rceil + c$ with high probability, the storage for z will be

$$\lceil \lg z \rceil = \lg \lg d + O(1) \leq \lg \lg n$$

2.4 Expected Output

The analysis so far begs the question: What exactly is the expected output, or $\mathbb{E}[2^z]$? Flajolet and Martin showed, with a proof that employed complex analysis, that $\mathbb{E}[2^z]$ rapidly tends towards d scaled by a specific constant:

Theorem 2.1 (Flajolet-Martin 85)**.** $E[2^z] \approx 0.77551d$.

This makes the PC algorithm eminently practical: Simply compute $E[2^z]$, then use divide it by 0.77551 (referred to as the "correction factor").

3 Improvement of the PC Algorithm

In [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan '04], the authors refine the PC algorithm to give a $(1 \pm \epsilon)$-approximation to d using storage $O(\epsilon^{-2} \lg(\frac{1}{\epsilon}) + \epsilon^{-2} \lg \lg n)$.

The idea is that, instead of tracking the single largest $r(i)$ value, we store r such that there are "sufficient many" elements i have $r(i) \geq r$. This ensures better convergence. To do this, we use two hash functions h (defined just like before) and a new random hash function

$$g : [n] \to [\epsilon^{-4} \lg^2 n]$$

That is, g maps each element to a random, shorter bitstring of length $\lg(\epsilon^{-4} \lg^2 n)$.

Unlike the original PC algorithm of Flajolet-Martin, the algorithm of Bar-Yossef et al. requires only *pairwise* independence in the hash functions (or *2-universal* hash functions), which is something we know how to provably construct.

The sketch maintained by the algorithm is a value z (just as before), as well as a set B containing all elements such that $r(i) \geq z$. However, storing an element i can be expensive, so instead, we store its short hash $g(i)$. The total space required will be

$$\begin{aligned} |B| \times \text{ [size in bits of each element in } B] &\leq \frac{c}{\epsilon^2} \times \left(\underbrace{\lg(\frac{1}{\epsilon}) + \lg \lg n}_{\text{space required for g(i)}} + \underbrace{\lg \lg n}_{\text{space required for r(i)}} \right) \\ &= O\left(\frac{1}{\epsilon^2} \times (\lg(\frac{1}{\epsilon}) + \lg \lg n) \right) \end{aligned}$$

The algorithm is as follows:

Algorithm 2 improved-PC Algorithm

```
z ← 0
for i in the stream do
  if r(i) ≥ z then
    B ← B ∪ {(g(i), r(i))}
    while |B| ≥ c/ε² do
      z ← z + 1
      remove all pairs (g(j), r(j)) such that r(j) < z
    end while
  end if
  z ← max(z, r(i))
end for
return |B| × 2^z        {as the estimate of the number of distinct element}
```

3.1 Analysis of the Algorithm

Roughly speaking, at any point, there are at most c/ϵ^2 elements i seen so far with $r(i) \geq z$. We claim that, when we randomly map each element (using g) to a universe $\left[\frac{c^2}{\epsilon^4}\lg^2 n\right]$, by the birthday paradox, the probability of two such elements i and j (which satisfy $r(i) \geq z$ and $r(j) \geq z$) to have the same hash value $g(i) = g(j)$ is much less than $\frac{1}{\lg n}$.

Proof. Assume we will need the size of the universe of g to be at least $|g|$ to achieve the collision rate at $\frac{1}{\lg n}$ for all the z values.

To see this, assume X_{ab} is the indicator of whether the a-th and b-th sample colliding, and $X = \sum_{1\leq a<b\leq \frac{c}{\epsilon^2}} X_{ab}$ is the number of pairwise collision after seeing $\frac{c}{\epsilon^2}$ elements with $r(i) \geq z$ for a particular value of z. Then

$$\begin{aligned} E[X] &= E\left[\sum_{1\leq a<b\leq \frac{c}{\epsilon^2}} X_{ab}\right] \\ &= \sum_{1\leq a<b\leq \frac{c}{\epsilon^2}} E\left[X_{ab}\right] \\ &= \sum_{1\leq a<b\leq \frac{c}{\epsilon^2}} Pr\left[g(a) = g(b)\right] \\ &= \binom{\frac{c}{\epsilon^2}}{2}\frac{1}{|g|} \end{aligned}$$

According to Markov inequality, we have:

$$Pr\left[X > t\right] \leq \frac{E[X]}{t}$$

if we want to the probability of a collision to be less than $\frac{1}{\lg^2 n}$ for one specific z value, we should set $t = 1$ and:

$$Pr\left[X > 1\right] \leq \frac{E[X]}{1} = \binom{\frac{c}{\epsilon^2}}{2}\frac{1}{|g|} \leq \frac{1}{\lg^2 n}$$

which will give us $|g| \geq \frac{C^2}{\epsilon^4} \lg^2 n$. (Notice that this holds for a particular z value).

Define X_i as the possible number of pairwise collision for the i-th z value. Then by union bound over all $\lg n$ values of z, we have:

$$\begin{aligned} Pr[\text{no collision for all possible } z \text{ value}] &= Pr[\max_i X_i > 1] \quad \text{(by union bound)} \\ &\leq \Pr[\bigcup_{i=1}^{\lg n} (X_i > 1)] \\ &\leq \lg n \times \frac{1}{\lg^2 n} \\ &= \frac{1}{\lg n} \end{aligned}$$

Thus, with high probability $(1 - \frac{1}{\lg n})$, g never creates collisions in B.

□

From here on, we'll assume that B stores i directly instead of using $g(i)$ to distinguish different i value (aka we assume that there are no-collision.)

Let $X_{l,j}$ be the indicator for the event $r(j) \geq l$. We know that $\mathbb{E}[X_{l,i}] = 2^{-l}$. Let $Y_l = \sum_{i \text{ in stream}} X_{l,i}$ be the number of elements i such that $r(i) \geq l$. Our estimate can therefore be written as $2^z|B| = 2^z Y_z$. We have:

$$\begin{aligned} \Pr\left[|2^z Y_z - d| > \epsilon d\right] &= \Pr\left[|Y_z - d2^{-z}| > \epsilon d 2^{-z}\right] \\ &= \Pr\left[|Y_z - \mathbb{E}[Y_z]| > \epsilon \mathbb{E}[Y_z]\right] \\ &\leq 2\exp\left(-\frac{\epsilon^2 \mathbb{E}[Y_z]}{3}\right) \end{aligned}$$

where we used the Chernoff bound in the last step.

There is a subtlety here: for the Chernoff bound to apply, $\mathbb{E}[Y_z]$ must be sufficiently large. To show this, we note that for all l,

$$\begin{aligned} \Pr[|Y_l - d2^{-l}| > \epsilon d 2^{-l}] &\leq 2\exp\left(-\frac{\epsilon^2 \mathbb{E}[Y_z]}{3}\right) \\ &\leq 2\exp\left(-\frac{\epsilon^2 d 2^{-l}}{3}\right) \end{aligned}$$

If $d2^{-l} \geq \frac{20}{\epsilon^2}$, then the Chernoff bound applies. In the next lecture we'll show that this is true with high probability, which allows us to conclude that $d2^{-z}$ is indeed close to its expectation.