CS 395T: Sublinear Algorithms Fall 2016

Lecture 2 — August 30, 2016

Prof. Eric Price *Scribes: Loc Hoang and Vasileios-Orestis Papadigenopoulos*

1 Overview

In the last lecture, we briefly introduced the three main areas that will be covered in the class along with examples.

- Data streams
- Property testing
- Compressed sensing

In this lecture, we discussed **concentration inequalities** and presented a sublinear algorithm for the **distinct elements** problem.

2 Concentration Inequalities

2.1 Markov's Inequality

Definition 1. *Let $X \geq 0$ be a random variable and $t > 0$.*

$$Pr[X \geq t] \leq \frac{E[X]}{t} \tag{1}$$

Proof. For $t \geq 0$, the following is straightforward.

$$E[X] \geq t \cdot Pr[X \geq t]$$

If we divide both sides by t, the result is *Markov's inequality.* □

2.2 Chebyshev's Inequality

Definition 2. *For **any** random variable X, $\mu = E[X]$, $\sigma^2 = Var[X] = E[(X - \mu)^2]$:*

$$Pr[|X - \mu| \geq t] \leq \frac{\sigma^2}{t^2} \qquad \forall t > 0 \tag{2}$$

Proof. Given $t > 0$:

$$Pr[|X - \mu| \geq t] = Pr[(X - \mu)^2 \geq t^2]$$

Since $(X - \mu)^2$ is itself a random variable, we can apply Markov's inequality.

$$Pr[(X - \mu)^2 \geq t^2] \leq \frac{E[(X - \mu)^2]}{t^2}$$

The numerator is simply variance. This proves *Chebyshev's Inequality.*

$$Pr[(X - \mu)^2 \geq t^2] \leq \frac{\sigma^2}{t^2}$$

□

2.3 Chernoff Bounds

Definition 3. *Let X be a random variable that is the sum of many **independent** variables , i.e.:*

$$X = \sum_{i=1}^{n} x_i, \text{ where } x_i \in \{0, 1\}$$

Let $\mu = E[X]$. In this case, for all $\varepsilon > 0$, the following bounds hold.

Upper Tail Bound

$$Pr[X \geq (1 + \varepsilon)\mu] \leq e^{-\frac{\varepsilon^2}{2+\varepsilon}\mu} \tag{3}$$

Lower Tail Bound

$$Pr[X \leq (1 - \varepsilon)\mu] \leq e^{-\frac{\varepsilon^2}{2}\mu} \tag{4}$$

Note that, apart from the aforementioned inequalities, Chernoff Bounds appear in many different forms in the literature. Generally, the choice of the Chernoff bound that gives "good" results is usually a problem-specific procedure. The proof for these Chernoff bounds will be covered in another lecture.

3 Distinct Elements

3.1 Problem Introduction

Consider a sequence (data stream) consisting of numbers from a universe $[n]$ (e.g. $1, 1, 4, 5, 200, 3, \cdots \in [n]$). Our goal is to estimate the number k of distinct elements in this sequence. Note that trying

to find the exact value of k is a non-trivial task in terms of space complexity. An easy approach would be to hash every element in the stream and count how many distinct elements appeared. However, this task would require $\mathcal{O}(n)$ space.

For this reason, the following algorithm is proposed that can approximate the value of k within a factor of $(1+\varepsilon)$ with probability $1-\delta$, running only in sublinear space.

We demonstrate here for $\varepsilon = 1$, or an approximation factor of 2.

3.2 A Relaxed Decision Problem

To estimate k, we first solve the following relaxed problem.

Define t to be some threshold number. **Is k larger than $2t$ or less than t?**

An algorithm that answers this question implies another algorithm to estimate k. This is accomplished by testing a number of different thresholds (e.g. $1, 2, 4, 8, 16, \ldots, 2^q, \ldots$) until we find an interval where k can lay. We will have then estimated k within a factor of 2.

Algorithm:

- Choose a random subset $S \subseteq [n]$ such that $Pr[x \in S] = \frac{1}{t}$, $\forall x \in [n]$.
- Record whether $\{S \cap \text{stream} \neq \emptyset\}$.

Let S be a subset of the universe $[n]$. For each element $i \in [n]$, let $Pr[i \in S] = \frac{1}{t}$.

Define Y to be the event where $\{S \cap \text{stream} \neq \emptyset\}$. We can determine the probability of Y.

$$\begin{aligned} Pr[Y] &= 1 - Pr[\bar{Y}] \\ &= 1 - (1 - \frac{1}{t})^k \end{aligned} \tag{5}$$

The last equality follows by the fact that the probability of Y not happening is the same the probability of all k elements falling outside of S.

Considering the cases that need to be distinguished, we can see the following.

$$Pr[Y|k \leq t] \leq 1 - (1 - \frac{1}{t})^t \approx 1 - \frac{1}{e} \approx 63.2\% \tag{6}$$

$$Pr[Y|k \geq 2t] \geq 1 - (1 - \frac{1}{t})^{2t} \geq 1 - \frac{1}{e^2} \approx 86.5\% \tag{7}$$

Given the gap between these values, we have managed to distinguish between a "more likely" and a "less likely" event. Therefore, if we run the same procedure n times independently, we get the following random variables $Y_1, Y_2, \ldots, Y_n$. We can do this for multiple ts in parallel as well. Since the gap between the 2 probabilities exists, it will be possible to determine around which t our k

exists within a factor of 2 since there will be a threshold where $k \leq t$ becomes $k \geq 2t$. We can use that change to determine an estimate of k.

If $n \geq \mathcal{O}(\log \frac{1}{\delta})$, we get the right answer with probability $1 - \delta$.

The total space we will use is $O(\log \frac{\log n}{\delta} \log n)$ in addition to the space used by the randomness requirement. $\log \frac{\log n}{\delta}$ is from the number of times we independently run a test for Y, and δ can be changed to trade off between space and accuracy. The $\log n$ comes from the number of ts we run in parallel.

3.3 Dealing with the Randomness Requirement

In the previous algorithm, we did not take into account the space complexity due to the use of randomness, i.e., the creation of the random set $S \subseteq [n]$. Generally, there are several ways to deal with the randomness requirement in an algorithm:

1. Ignore the issue completely.
    - Using the randomness of the input data, if it is considered sufficient.
    - Using cryptographic hash functions.
    - Developing algorithms in the *Random Oracle Model*.
2. Psuedo-random number generators.
3. Use a limited dependence on randomness.

Returning to the discrete elements problem, we are interested in third aforementioned option. Specifically, we will be using **pairwise independent hashes** to facilitate/limit randomness.

Definition 4. ***Pairwise Hash Functions.*** *Let $\mathcal{H}$ be a family of functions of the type $h_i[n] \to [B]$. We say that family $\mathcal{H}$ is* pairwise independent *if for every $x, y \in [n]$ such that $x \neq y$ and for every $\alpha, \beta \in [B]$ it is the case that*

$$Pr_{h \in \mathcal{H}}[h(x) = \alpha \cap h(y) = \beta] = \frac{1}{B^2}$$

For example, let B a prime number greater than n. The following family of functions is pairwise independent:

$$\mathcal{H} = \{h_{c,d}(x) = cx + d \mod B | c, d \in [B]\}$$

3.4 Distinct Elements, Revisited

Now, using the previous hash function for the creation of S, we have that

$$Pr[x \in S] = Pr[h(x) = 0] = \frac{1}{B}$$

Note that, for any sequence of events $A_1, A_2, \dots A_n$, we can extract an upper and a lower bound to the probability of their union using the following relations:

$$\sum_i Pr[A_i] - \sum_{i<j} Pr[A_i \cap A_j] \leq Pr[A_1 \cup A_2 \cup \cdots \cup A_n] \leq \sum_i Pr[A_i]$$

Therefore, we can see that

$$\frac{k}{B} \geq Pr[S \cap \text{stream} \neq \emptyset] \geq \frac{k}{B} - \sum_{i<j} Pr[x_i \in S \cap x_j \in S]$$

The first inquality holds since the stream has k items with each item having a $\frac{1}{B}$ chance of appearing in the subset S.

We also can see that

$$\begin{aligned} Pr[S \cap \text{stream} \neq \emptyset] &\geq \frac{k}{B} - \sum_{i<j} Pr[x \in S \cap y \in S] \\ &\geq \frac{k}{B} - \frac{k(k-1)}{2} Pr[x \in S \cap y \in S] \\ &\geq \frac{k}{B}(1 - \frac{k-1}{2B}) \end{aligned}$$

The final transition above happens since the probability of $x, y \in S$ is $\frac{1}{B^2}$.

Given this, we can see that

$$Pr[Y|k \leq t] \leq \frac{t}{B}$$

$$Pr[Y|k \geq 2t] \geq \frac{2t}{B}(1 - \frac{t}{B})$$

where Y is $\{S \cap \text{stream} \neq \emptyset\}$.

Since all we need is the second value to be greater than the first, choosing $B = 10t$, we get $Pr[Y|k \leq t] \leq 0.1$ and $Pr[Y|k \geq 2t] \geq 0.18$. Therefore, if we take $\mathcal{O}(\log(\frac{1}{\delta}))$ repetitions using different hash functions from the pairwise independent family, with probability $1 - \delta$ we can distinguish between the 2 cases and proceed as we discussed previously.

In conclusion, we can see that the total space we use for the execution of our algorithm using hash functions is $\mathcal{O}(\log n \log \frac{\log n}{\delta} \log n)$, where $\mathcal{O}(\log n)$ are the times needed to test different values of t, $\mathcal{O}(\log \frac{\log n}{\delta})$ is the cost of the trials, and $\mathcal{O}(\log n)$ the cost of hashing.

3.5 A General Proof

It is possible to adapt this proof for $\varepsilon = 1$ to the more general case as was discussed at the end of the lecture.