CS-621 Theory Gems November 15, 2012

Lecture 17

Lecturer: Aleksander Mądry *Scribes: Mani Bastani Parizi and Christos Kalaitzis*

1 Introduction

We continue our treatment of streaming algorithms. Last time, we developed streaming algorithm for distinct elements problem. Today, we discuss in more detail one of the pieces that we needed in that construction – k-wise independent hash functions. Then, we introduce the L_p-norm estimation problem and design an algorithm for the L_2-norm estimation.

2 k-wise Independent Hash Functions

As we mentioned in the last lecture, one of the core components of our algorithm for distinct elements problem is a space-efficient construction of 2-wise independent functions. Formally, a *k-wise independent hash function* $f : [m] \to [T]$ is a randomized function that provides the guarantee that, for any k distinct elements $j_1, \ldots, j_k \in [m]$ and any k possible values $t_1, \ldots, t_k \in [T]$, the probability (over the randomness in the construction of the function f) that $f(j_i) = t_i$, for all i, is exactly $\frac{1}{T^k}$.

It is easy to see that every fully random function is k-wise independent hash function, for any k. But it turns out that there are constructions of k-wise independent hash functions that can be stored using relatively small space (and thus are not fully random). So, if our algorithm is ok with using a hash function that provides only a k-wise independence guarantee for some (small) k – as was, for example, the case in our algorithm for distinct elements problem – we can exploit it to get an improved space complexity.

We proceed now to presenting a particularly simple construction of 2-wise independent hash function. To this end, let us assume that T is prime and consider a function $h : [m] \to [T]$ given by

$$h(j) = a \cdot j + b \mod T \tag{1}$$

where a and b are chosen independently and uniformly from $[T]$.

It is not hard to see that this hash function is indeed a 2-wise independent function.

Lemma 1 *If T is prime, the hash function h (as defined by Equation (1)) is 2-wise independent.*

Proof Consider some $j, j' \in [m]$ with $j \neq j'$. We would like to compute the probability (over the choice of a and b in definition of h) that $h(j) = s$ and $h(j') = s'$ for some fixed $s, s' \in [T]$. By definition of h, this corresponds to the event that

$$a \cdot j + b = s \mod T \quad \text{and} \quad a \cdot j' + b = s' \mod T.$$

We can view the above equations as a linear system over $\mathbb{Z}_T$ with a and b being the variables, j, j' being the coefficients, and s and s' giving the constraints

Note that if T is prime then $\mathbb{Z}_T$ is a field. Furthermore, the determinant of this linear system is non-zero (as $j \neq j'$). So, this implies that there is a unique solution a^* and b^* that makes both of these equations satisfied. As a result, by the fact that a and b are chosen independently and uniformly from $[T]$,

$$\Pr[h(j) = s, h(j') = s'] = \Pr[a = a^*, b = b^*] = \left(\frac{1}{T^2}\right),$$

as desired. ■

Clearly, such a function requires only $O(\log T)$ space to store as we just need to memorize a and b.[1] Also, one can easily extend this construction to obtain a k-wise independent hash function for arbitrary k, with $O(k \log T)$ space complexity, by just making h to be given by a polynomial of degree k with coefficient chosen uniformly and independently from $[T]$.

Finally, to deal with the case when T is not prime, one can just choose some prime number T' that is larger than T and then define h to be

$$h(j) = (a \cdot j + b \mod T') \mod T.$$

If T' is large enough, one can see that the resulting hash function is close to being 2-wise independent.

3 L_p-norm Estimation

One of the most fundamental problem (or, rather, a family of problems) in streaming algorithms is the estimation of some L_p-norm of the element frequencies. Formally, by an (ε, δ)-approximation to the *L_p-norm estimation* problem is a streaming algorithm that, with probability at least $1 - \delta$, returns an estimate that is within $(1 + \varepsilon)$ multiplicative error of the L_p-norm[2] $\|x\|_p$ of the vector x, where

$$\|x\|_p := \left(\sum_{j \in [m]} |x_j|^p \right)^{1/p}$$

and we recall that x_j is the number of occurrences of the element j in the stream $\boldsymbol{y} = (y_1, \ldots, y_n)$.

One reason why L_p-norm estimation is important stems from the fact that – depending on the value of p – it captures many natural problems. For example:

- $\|x\|_0$ measures the number of non-zero coordinates of x. Hence, L^0-norm estimation is the distinct elements problem we just studied.
- $\|x\|_1$ is simply the length of the stream. It can trivially be computed exactly using $O(\log n)$ space.[3]
- $\|x\|_\infty$ measures the frequency of the most frequent element. So, L_∞-norm estimation captures computing of the most frequently occurring element.

Now, given the fundamental nature of this problem, there was a lot of research on this front. It resulted in quite good understanding of its complexity. In particular, it turns out that for $p \in [0, 2]$ the problem can be solved in $O\left((\frac{1}{\varepsilon})^{O(1)} \log^{O(1)} n\right)$ space. However, for $p \in (2, \infty]$ it is now known that the space complexity of L_p-norm estimation is $\Theta\left(m^{2-\frac{2}{p}} \log^{O(1)} n\right)$. As a result, somewhat disappointingly, the L_∞-norm estimation problem (even when we allow approximation) requires $\Omega(m)$ space.

4 L_2-norm Estimation

In the coming lectures, we will see a variety of results on L_p-norm estimation. Here, we start with the case of $p = 2$.[4] To simplify the presentation, we will focus here on estimating the ℓ_2^2-norm instead of

[1] Note that in our algorithm for distinct elements problem from last lecture, we had $T \leq n$ and thus this function can indeed be stored in $O(\log n)$ space as claimed there.

[2] Given that the vector x is always finite-dimensional, the mathematically precise term should be the ℓ_p-norm instead of L_p-norm. But, for historic reasons, the latter name is used.

[3] Note that this true only for the basic streaming model, in which we only allow insertions of elements. L_1-norm estimation in the turnstile model, i.e., when also element deletions are allowed, is not that trivial.

[4] Although L_2-norm estimation might not seem to be a very natural problem, it actually has some compelling applications (e.g., in query answering in relational databases) and, even more importantly, it is often used as a building block in solutions for other streaming problems.

the ℓ_2-norm. It is easy to see that from the point of view of $(1+\varepsilon)$-approximation, estimating ℓ_2- and ℓ_2^2-norm is equivalent - after all, one quantity is just a square of the other.

Similarly to the case of distinct elements problem. We will start by presenting a simple algorithm – let's call it, again, Algorithm A – that encapsulates the core of our approach, but has large probability of failure. Then, we develop a way of using this simple algorithm as a black-box to obtain the desired correctness guarantee.

Our Algorithm A works as follows:

- For every element $j \in [m]$, choose r_j to be either 1 or -1 independently and equiprobably.
- Make a pass over the stream and compute the following estimator

$$Z = \sum_{j \in [m]} r_j x_j.$$

- At the end, output Z^2 as the answer.

It is not hard to see that the estimator Z can be easily maintained as we pass over the stream (provided we know all r_js).

Now, to analyze the quality of the returned estimate, let us take a look at its expected value $\mathbb{E}[Z^2]$. We have that

$$\mathbb{E}[Z^2] = \mathbb{E}[(\sum_{j \in [m]} r_j x_j)^2] = \sum_{j_1, j_2 \in [m]} \mathbb{E}[r_{j_1} x_{j_1} r_{j_2} x_{j_2}].$$

Observe that

$$\mathbb{E}[r_{j_1} r_{j_2}] = \begin{cases} 1 & j_1 = j_2 \\ 0 & \text{otherwise,} \end{cases}$$

as when $j_1 \neq j_2$, r_{j_1} and r_{j_2} are independent and thus $\mathbb{E}[r_{j_1} r_{j_2}] = \mathbb{E}[r_{j_1}]\mathbb{E}[r_{j_2}] = 0$.

This gives us that

$$\mathbb{E}[Z^2] = \sum_{j \in [m]} x_j^2 = \|x\|_2^2,$$

so our estimator has the correct value in expectation.

Of course, the fact that expectation is correct does not yet yield anything about the correctness of the algorithm, as this does not provide any bound on the error probability. So, to get a handle on the concentration of the estimator Z^2 around its expected value, we will analyze its variance.

Recall that the variance $\mathrm{Var}[X]$ of a random variable X is defined as $\mathrm{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. We have that

$$\mathrm{Var}[Z^2] \leq \mathbb{E}[Z^4] = \sum_{j_1, j_2, j_3, j_4 \in [m]} \mathbb{E}[r_{j_1} r_{j_2} r_{j_3} r_{j_4}] x_{j_1} x_{j_2} x_{j_3} x_{j_4}.$$

Observe that, as all r_{j_l}s are independent and their expectations are 0, we have

$$\mathbb{E}[r_{j_1} r_{j_2} r_{j_3} r_{j_4}] = \begin{cases} 0, & \text{if some } j \text{ appears exactly one or three times} \\ 1, & \text{otherwise.} \end{cases}$$

This, in turn, means that

$$\mathrm{Var}[Z^2] \leq 6 \sum_{j_1, j_2 \in [m]} x_{j_1}^2 x_{j_2}^2 = 6\|x\|_2^4.$$

Now, a natural way of bounding the concentration of a variable based on its variance is using Chebyshev's inequality.

Theorem 2 (Chebyshev's inequality) *For any $\gamma > 0$, $\Pr[|X - \mathbb{E}[X]| \geq \gamma] \leq \frac{\mathrm{Var}[X]}{\gamma^2}$*

As we are interested in obtaining an $(1+\varepsilon)$-approximation, we need to set $\gamma = \varepsilon\|x\|_2^2$. This gives us that the probability of failure of Algorithm A is at most

$$\Pr\left[|Z^2 - \mathbb{E}[Z^2]| \geq \varepsilon\|x\|_2^2\right] \leq \frac{\mathrm{Var}[Z^2]}{\varepsilon^2\|x\|_2^4} \leq \frac{6\|x\|_2^4}{\varepsilon^2\|x\|_2^4} = \frac{6}{\varepsilon^2}.$$

Unfortunately, as $\frac{6}{\varepsilon^2}$ can be bigger than 1 (let alone δ) even for moderate values of ε, this bound is not strong enough to provide any meaningful guarantee. Therefore, similarly to the case of our algorithm for distinct elements problem, we need to develop a way of boosting the probability of success. To this end, consider the following algorithm (let's call it Algorithm B):

- Run $k = \frac{6}{\varepsilon^2\delta}$ (independent) instances of the Algorithm A in parallel.
- At the end, let $Z_1^2, \ldots, Z_k^2$ be the estimators returned by these k instances of Algorithm A, output the average $Z^* = \frac{1}{k}\sum_j Z_j^2$ of these estimators as the answer.

Clearly, the expectation of Z^* is exactly the same as the expectation of Z^2, i.e.,

$$\mathbb{E}[Z^*] = \frac{1}{k}\sum_j \mathbb{E}[Z_j^2] = \|x\|_2^2 = \mathbb{E}[Z^2].$$

Now, the crucial difference is that the variance of Z^* is smaller than the one of Z^2. Namely, as all Z_j^2s are independent, an elementary calculation shows that

$$\mathrm{Var}[Z^*] = \frac{\mathrm{Var}[Z^2]}{k} \leq \frac{6\|x\|_2^4}{k}.$$

So, applying the Chebyshev's inequality to Z^* (instead of Z^2) with $\gamma = \varepsilon\|x\|_2^2$, we get

$$\Pr\left[|Z^* - \mathbb{E}[Z^*]| \geq \varepsilon\|x\|_2^2\right] \leq \frac{\mathrm{Var}[Z^*]}{\varepsilon^2\|x\|_2^4} \leq \frac{6\|x\|_2^4}{k\varepsilon^2\|x\|_2^4} = \frac{6}{k\varepsilon^2} = \delta,$$

which gives us the desired failure probability bound.

Now, we proceed to analyzing the space complexity of our algorithm. Notice that if we ignore the space needed to store our selection of all the r_j, Algorithm A needs to maintain only one number and thus its space complexity is $O(\log n)$. Therefore, the space required by Algorithm B is just

$$O(k \log n) = O\left(\frac{\log n}{\epsilon^2\delta}\right).$$

So, it remains to discuss the space required to store the r_js in Algorithm A. To this end, we will again model our choice of r_js as choosing a hash function $h : [m] \to \{-1, 1\}$ and then setting $r_j = h(j)$. Now, the crucial thing to notice is that our analysis of correctness of Algorithm A depends only on 4-order moments $\mathbb{E}[r_{j_1} r_{j_2} r_{j_3} r_{j_4}]$ of r_js behaving properly for all $(r_{j_1} r_{j_2} r_{j_3} r_{j_4}) \in [m]^4$. Therefore, it suffices that our hash function h is just 4-wise independent to make the whole analysis go through.

As we discussed earlier, such a 4-wise independent hash function can be stored in only $O(1)$ space (as $T = 2$ here), thus the overall space complexity of Algorithm A is still $O(\log n)$. As a result, the Algorithm B indeed provides an (ε, δ)-approximation to the L_2-norm estimation problem in space $O(\frac{\log n}{\varepsilon^2\delta})$. (Note that, in contrast to our algorithm for distinct elements problem, the space complexity dependence on $\frac{1}{\delta}$ is only polynomial and not logarithmic.)