CSE 521: Design and Analysis of Algorithms I **Spring 2016**

Lecture 3: Streaming Algorithms

Lecturer: Shayan Oveis Gharan *April 4th* *Scribe: Antoine Bosselut*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

In this lecture, first we go over an application of the Hoeffding bound in discrepancy theory, then we go over basic families of streaming algorithms.

3.1 Set Balancing

Given a matrix $A \in \{0,1\}^{n\times n}$, we want to find a vector $b \in \{\pm 1\}^n$ minimizing $\|Ab\|_\infty$. Recall that for a vector $v \in \mathbb{R}^n$,

$$\|b\|_\infty = \max_{1\le i\le n} |b_i|.$$

This problem is one of the most basic problems in discrepancy theory. The field of discrepancy theory has many applications in various areas of computer science including Approximation algorithms, communication complexity, machine learning and optimization. We refer interested students to [Cha00]. Here, we prove the following theorem.

Theorem 3.1. *Let b be a uniformly random vector with $+1$ and -1 coordinates. Then,*

$$\mathbb{P}\left[\|Ab\|_\infty \le O(\sqrt{n\log n})\right] \ge 1 - 1/n.$$

Thea above bound is the best possible bound if we want a high probability result. However, there is an efficient algorithm which returns a vector $b \in \{\pm\}^n$ such that $\|Ab\|_\infty \le O(\sqrt{n})$ [Spe85; Ban10; LM15].

Proof. Let $a_1, a_2, \ldots, a_n$ be the rows of A, i.e.,

$$A = \begin{pmatrix} \text{---}\ a_1\ \text{---} \\ \text{---}\ a_2\ \text{---} \\ \vdots \\ \text{---}\ a_n\ \text{---} \end{pmatrix}$$

Then, for any $1 \le i \le n$,

$$(Ab_i) = \langle a_i, b\rangle = \sum_{j=1}^n a_{i,j} b_j.$$

Therefore,

$$\mathbb{E}\left[\langle a_i, b\rangle\right] = \sum_{j=1}^n \mathbb{E}\left[a_{ij}\cdot b_j\right] = \sum_{j=1}^n a_{i,j}\mathbb{E}\left[b_j\right] = 0,$$

where the last equality uses that each b_j is zero in expectation. So, all we need to do is to show that $\langle a_i, b\rangle$ is strongly concentrated around its expectation. We use the Hoeffding inequality to prove this.

Fix some $1 \le i \le n$. First, observe that if $a_{i,j} = 0$, b_j doesn't matter. For any $1 \le j \le n$ let

$$X_j = \begin{cases} b_j & \text{if } a_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Let $X = X_1 + \cdots + X_n$. Observe that the distribution of X is similar to a random walk process of length $\|a_i\|$. So, similar to the previous lecture we can use the Hoeffding bound to upper bound $|X|$. For $\epsilon = \sqrt{16\|a_i\|_1 \log(n)}$, we have

$$\mathbb{P}\left[|X| \ge \epsilon\right] \le \exp\left(-\frac{2\epsilon^2}{\sum_{j=1}^n a_{i,j}(+1-(-1))^2}\right) = \exp\left(-\frac{2\epsilon^2}{4\|a_i\|_1}\right) = e^{-2\log n} = \frac{1}{n^2}.$$

In other words,

$$\mathbb{P}\left[|\langle a_i, b\rangle \ge 4\sqrt{n\log n}\right] \le \frac{1}{n^2}.$$

This says that for any $1 \le i \le n$, the i-th coordinate of Ab is at most $O(\|a_i\|_1 \log n)$ with a very high probability.

Now, let $\mathcal{E}_i$ be an event that indicates $< a_i, b > \ge \sqrt{4\|a_i\|_1 \log n}$. We are interested in the event that none of the $\mathcal{E}_i$'s occur. Unfortunately, we do not know how these events correlate with one another. The only thing that we know is that each of them occurs with a tiny probability.

The idea is to use the union bound. It says that for any family of events $\mathcal{E}_1, \ldots, \mathcal{E}_n$,

$$\mathbb{P}\left[\cup_{i=1}^n \mathcal{E}_i\right] \le \sum_{i=1}^n \mathbb{P}\left[\mathcal{E}_i\right].$$

So, in our case,

$$\mathbb{P}\left[\cap \neg\mathcal{E}_i\right] = 1 - \mathbb{P}\left[\cup_{i=1}^n \mathcal{E}_i\right] \ge 1 - \sum_{i=1}^n \mathbb{P}\left[\mathcal{E}_i\right] \ge 1 - 1/n.$$

In other words, $\mathbb{P}\left[\forall_i \langle a_i, b\rangle \le \sqrt{16n\log n}\right] \ge 1 - \frac{1}{n}$ □

3.2 Streaming

In the next two weeks we will see several applications of randomization and concentration bounds in algorithm design. Our first family of applications is in designing streaming algorithms.

Suppose we are given a sequence of objects $x_1, x_2, \ldots, x_m$ from a large universe $\mathcal{U}$. We want to design an algorithm that reads the elements of this sequence one by one and it and answers specific set of queries related to this sequence. The main restriction is that our algorithm is only allowed to use a small amount of memory, ideally only poly-logarithmic in the length of the sequence and the size of the universe $\mathcal{U}$.

In this lecture and the next we study perhaps the simplest set of queries one might ask about the sequence namely the 0-th and the second moments. Also, for the sake of simplicity we assume that $\mathcal{U} = [n]$ is simply the set of the integers from $0, 1, \ldots, n-1$. Let F_0 be the number of distinct elements in the sequence. Also, for any $k \ge 1$ let $F_k = \sum_i \{\#x_j = i\}^k$ be the k-th moment of the number of times that each element occurs in the sequence. In particular, F_∞ is the maximum number of times an element appears in the sequence. The following theorems are proved in [AMS96].

Theorem 3.2. *For any sequence $x_1, \ldots, x_m$ and $\epsilon, \delta > 0$, F_0 and F_2 can be approximated within a $1 \pm \epsilon$ factor with probability $1 - \delta$ in space $O\big((\log n + \log m) \log \frac{1}{\delta}/\epsilon^2\big)$.*

Although the F_0 and F_2 can be approximated with only logarithmic amount of memory, these are the only special cases. For all higher moments we need at least a polynomial amount of memory in n even to return approximate answers.

Theorem 3.3 ([AMS96])**.** *If a (randomized) algorithm approximates F_∞ within $1 \pm .2$ factor with probability $\frac{2}{3}$, it needs at least $\Omega(\min(m, n))$ memory.*

Theorem 3.4 ([IW05])**.** *For finite $k \geq 2$ the best memory one needs to approximate F_k is $\Theta(n^{1-\frac{2}{k}})$*

In the rest of this lecture we prove the F_0 case of Theorem 3.2. In the next lecture we prove the F_2 case. So, fix a sequence $x_1, \ldots, x_m$. We want to design an algorithm that maintain a *sketch* of the sequence such that at the end of the input it can return the number of distinct elements within a $1 \pm \epsilon$ factor with $1 - \delta$ probability. Note that our algorithm needs to work in the worst case. That is we cannot assume the sequence $x_1, \ldots, x_m$ is a random sequence generated from the set $[n]$. In fact, if this was a random sequence, for a large enough n, with high probability all elements of the sequence were disjoint and we could simply return m as the answer. The latter fact simply follows from the birthday paradox that we studied in the last lecture.

First Attempt. Consider the following simple algorithm. Choose each element of the sequence with probability proportional to $\frac{\log(n)}{\epsilon^2 m}$ and return the number of distinct element in the sampled subsequence time $\epsilon^2 m / \log(n)$. In other words, we want to reduce our "big data" problem to a small problem that we can solve exactly with the limited available memory. Unfortunately, this simple idea fails for the following adversarially chosen sequence: $\underbrace{1, 1, \ldots, 1}_{m-k \text{ times}}, 2, 3, \ldots, k-1$. For such a sequence, with high probability we only see 1's in the sample.

Idea. Let us first solve a simpler problem. Suppose we are given an integer k; if $F_0 < k$ we have to return no and if $F_0 > 2k$ we have to return yes. For all values of F_0 we can return yes/no arbitrarily. If we can solve this problem using a small amount of memory that we can use it to estimate F_0 simply by running the procedure simultaneously for all powers of 2 which are less than n. This would give a 2-approximation. To get the error probability down to $1 + \epsilon$ we need to distinguish the cases $F_0 < k$ and $F_0 > (1 + \epsilon)k$ which can be done by similar ideas.

Let $2k \leq B \leq 4k$ be an integer. Let $\mathcal{H} = \{h : [n] \to [B]\}$ be the family of all functions that map $[n]$ to $[B]$. Observe that a uniformly random function $h \sim H$ maps each integer in $[n]$ to a uniformly and independently random integer in $[B]$. Now, consider the following algorithm: Return yes if there is an x_i in the sequence for which $h(x_i) = 0$ and return no otherwise. Let us analyze this algorithm. Since $h(x_i)$ is chosen uniformly at random, for each i,

$$\mathbb{P}_{h \sim \mathcal{H}} [h(x_i) = 0] = \frac{1}{B}$$

Since there are F_0 distinct number in the sequence and the value of $h(.)$ for each of these numbers is chosen independently,

$$\mathbb{P}_{h \sim \mathcal{H}} [\forall i : h(x_i) \neq 0] = (1 - \frac{1}{B})^{F_0}.$$

Therefore,

$$\mathbb{P}_{h \sim \mathcal{H}} [\exists i : h(x_i) = 0] = 1 - (1 - \frac{1}{B})^{F_0}.$$

Now, let us consider the two cases $F_0 < k$ and $F_0 > 2k$. We use p_1 to denote $\mathbb{P}_{h \sim \mathcal{H}} [\exists i : h(x_i) = 0]$ in case 1 and p_2 for the similar quantity in case 2.

Case 1: $F_0 < k$**.** In this case we can write, $p_1 \leq (1 - \frac{1}{B})^k \approx 1 - e^{-\frac{1}{2}}$ if $B = 2k$

Case 2: $F_0 > 2k$ However, here we have $p_2 \geq (1 - \frac{1}{B})^{2k} \approx 1 - e^{-1}$ if $B = 2k$

Observe that $p_2 - p_1 \geq 0.2$, i.e., there is a constant gap between two cases. Of course, we may be unlucky and even if $F_0 < k$ we get a number x_i in the sequence for which $h(x_i) = 0$, so we return yes incorrectly. So, the naive algorithm mentioned above fails.

To get around this issue all we need to do is to estimate $\mathbb{P}_{h\sim\mathcal{H}}[\exists i : h(x_i) = 0]$ with an error better than $\alpha = |(1 - e^{-1/2}) - (1 - 1/e)|$. By the Hoeffding bound all we need is $O(\log(1/\delta)/\alpha^2)$ independent samples of p. So, here is the modified algorithm: Let $r = O(\log(1/\delta)/\alpha^2)$. Let $h_1, \ldots, h_r : [n] \to [B]$ be r functions chosen independently from $\mathcal{H}$. For each h_i let $Y_i = \mathbb{I}[\exists j : h(x_j) = 0]$. If $\frac{1}{r}\sum_i Y_i$ is closer to $(1-1/B)^k$ return no and otherwise return yes.

By the Hoeffding bound, the above algorithm returns the correct answer with probability $1 - \delta$. Since α is constant we only need $O(\log(1/\delta)$ hash functions. The only caveat is that we need a large amount of space to store the hash functions $h_1, \ldots, h_r$. Since $\mathcal{H}$ has B^n many functions we need $O(n \log(B))$ memory to store any function h_i. Note that h_i's cannot be any predefined hash function as you typically see in programming languages. Such a function may map all of the numbers $x_1, \ldots, x_m$ in the sequence to the same number in $[B]$ in the worst case. Indeed randomness is necessary for the algorithm to work in the worst case. It turns out most of the above analysis can be done even if $\mathcal{H}$ is not the family of all functions $h : [n] \to [B]$. More precisely most of the analysis works out even if $h(i)$ for integers $i \in [n]$ is not truly independent but just pairwise independent. In the rest of this lecture we prove this statement. In the next lecture we see how we can construct a pairwise independent hash function using a family $\mathcal{H}$ with only n^2 many functions. A random function from such a family can be described in $O(\log n)$ bits.

3.2.1 Pairwise Independent Hash Functions

As before, let $\mathcal{H} = \{h : [n] \to [B]\}$ We say $\mathcal{H}$ is a family of pairwise independent hash functions if

$$\forall x \neq y; c, d \in [B], \mathbb{P}[h(x) = c, h(y) = d] = \frac{1}{B^2}.$$

Let $\mathcal{H}^*$ be a family of pairwise independent hash functions. Without loss of generality assume that the first F_0 numbers of the sequence are all distinct, i.e., $x_1, x_2, ..., x_{F_0}$ are distinct integers in $[n]$. This is a valid assumption because our algorithm is invariant under the ordering of the elements of the sequence. Now, let us consider the two cases.

Case 1: $F_0 < k$**.** We use the union bound to upper bound p_1.

$$p_1 = \mathbb{P}_{h\sim\mathcal{H}^*}\left[\bigcup_{i=1}^{F_0} h(x_i) = 0\right] \leq \sum_{i}^{F_0} \mathbb{P}_{h\sim\mathcal{H}^*}[h(x_i) = 0] = \frac{F_0}{B} \leq \frac{k}{B}$$

In the last equality we use that since $\mathcal{H}$ is a pairwise independent hash function, for any integer i, $\mathbb{P}[h(i) = 0] = 1/B$, and in the last inequality we use that $F_0 < k$.

Case 2: $F_0 > 2k$**.** In this case we need to lower bound p and show that it has a constant gap with k/B. First, recall the inclusion-exclusion principle, for any set of events, $\mathcal{E}_1, \ldots, \mathcal{E}_n$,

$$\mathbb{P}[\cup_i \mathcal{E}_i] = \sum_i \mathbb{P}[\mathcal{E}_i] - \sum_{i<j} \mathbb{P}[\mathcal{E}_i \cap \mathcal{E}_j] + \ldots$$

If we cut the RHS at odd terms we get upper bound on the LHS and if we cut it at even terms we get a lower bound; in particular, we can write

$$\sum_i \mathbb{P}\left[\mathcal{E}_i\right] - \sum_{i<j} \mathbb{P}\left[\mathcal{E}_i \cap \mathcal{E}_j\right] \leq \mathbb{P}\left[\cup_i \mathcal{E}_i\right] \leq \sum_i \mathbb{P}\left[\mathcal{E}_i\right]$$

Using the left inequality we can write,

$$\begin{aligned} p_2 = \mathbb{P}_{h\sim\mathcal{H}^*}\left[\cup_{i=1}^{F_0} h(x_i) = 0\right] &\geq \mathbb{P}_{h\sim\mathcal{H}^*}\left[\cup_{i=1}^{2k} h(x_i) = 0\right] \\ &\geq \sum_i^{F_0} \mathbb{P}_{h\sim\mathcal{H}^*}\left[h(x_i) = 0\right] - \sum_{1\leq i<j\leq F_0} \mathbb{P}\left[h(x_i) = h(x_j) = 0\right] \\ &= \frac{2k}{B} - \frac{\binom{2k}{2}}{B^2} \end{aligned}$$

where the first inequality uses that $F_0 > 2k$ and the equality uses that $\mathcal{H}^*$ is a family of pairwise independent hash functions.

Therefore,

$$p_2 - p_1 \geq \left(\frac{2k}{B} - \frac{k(2k-1)}{B^2}\right) - \frac{k}{B} \geq \frac{k}{B}(1 - 2k/B) \geq 1/8,$$

for $B = 4k$. Since there is a constant gap between p_1 and p_2, it is sufficient to use $r = O(\log \frac{1}{\delta}/\alpha^2)$ for $\alpha = 1/8$ independent functions chosen from $\mathcal{H}^*$ to estimate $\mathbb{P}_{h\sim\mathcal{H}^*}\left[existsi : h(x_i) = 0\right]$ within $1/16$ additive error. Since we need $O(\log n)$ bits to store each function from $\mathcal{H}^*$ with a $O(\log(1/\delta)\log(n))$ space we can test if $F_0 < k$ or $F_0 > 2k$. Using similar ideas we can test if $F_0 < k$ or $F_0 > (1+\epsilon)k$ in space $O(\log(1/\delta)\log(n)/\epsilon^2)$. Since there are $\log_{1+\epsilon} n$ possibilities for k, we can estimate F_0 within factor $1+\epsilon$ using only $O(\log(1/\delta)\log(n)\log_{1+\epsilon}(n)/\epsilon^2$. Note that the space dependency mentioned in Theorem 3.2 is slightly better than this. The reason is that [**AMS**] uses a different idea to estimate F_0 known as minhash.

References

[AMS96] N. Alon, Y. Matias, and M. Szegedy. "The space complexity of approximating the frequency moments". In: *STOCw*. ACM. 1996, pp. 20–29 (cit. on pp. 3-2, 3-3).

[Ban10] N. Bansal. "Constructive algorithms for discrepancy minimization". In: *FOCS*. IEEE. 2010, pp. 3–10 (cit. on p. 3-1).

[Cha00] B. Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, 2000 (cit. on p. 3-1).

[IW05] P. Indyk and D. Woodruff. "Optimal approximations of the frequency moments of data streams". In: *STOC*. ACM. 2005, pp. 202–208 (cit. on p. 3-3).

[LM15] S. Lovett and R. Meka. "Constructive Discrepancy Minimization by Walking on the Edges". In: *SIAM Journal on Computing* 44.5 (2015), pp. 1573–1582 (cit. on p. 3-1).

[Spe85] J. Spencer. "Six standard deviations suffice". In: *Transactions of the American Mathematical Society* 289.2 (1985), pp. 679–706 (cit. on p. 3-1).