CSE 525: Randomized Algorithms **Spring 2025**

Lecture 4: Strong Concentration Bounds

Lecturer: Shayan Oveis Gharan *04/10/2025*

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

We have seen how knowledge of the variance of a random variable X can be used to control deviation of X from its mean. This is the heart of the second moment method. But often we can control even higher moments, and this allows us to obtain much stronger concentration properties. A prototypical example is when $X_1, X_2, \ldots, X_n$ is a family of independent (but not necessarily identically distributed) $\{0,1\}$ random variables and $X = X_1 + X_2 + \cdots + X_n$. Let $p_i = \mathbb{E}[X_i]$ and define $\mu = \mathbb{E}[X] = \sum_{i=1}^n p_1 + p_2 + \cdots + p_n$. In that case, we have the following multiplicative form of the "Chernoff bound".

Theorem 6.1 (Multiplicative Chernoff bound)**.** . *For every $\delta \geq 0$, it holds that*

$$\mathbb{P}[X \geq (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu .$$

and

$$\mathbb{P}[X < (1-\delta\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu$$

Consequently,

$$\mathbb{P}[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/(2+\delta}, \mathbb{P}[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$$

Proof. Let t be a parameter that we choose later.

$$\mathbb{P}[X \geq (1+\delta)\mu] = \mathbb{P}\left[e^{tX} \geq e^{t(1+\delta)\mu}\right] \underset{\text{Markov's Inequality}}{\leq} \frac{\mathbb{E}\left[e^{tX}\right]}{e^{t(1+\delta)\mu}}. \tag{6.1}$$

The first inequality uses that the exponential function is a monotone function.

Now, we can write

$$\mathbb{E}\left[e^{tX}\right] = \mathbb{E}\left[e^{t\sum_i X_i}\right] = \mathbb{E}\left[\prod_{i=1}^n e^{tX_i}\right] \underset{\text{independence}}{=} \prod_{i=1}^n \mathbb{E}\left[e^{tX_i}\right].$$

Now, observe that

$$\mathbb{E}\left[e^{tX}\right] = p_i e^t + (1-p_i) = 1 + p_i(e^t - 1) \underset{1+x\leq e^x}{\leq} e^{p_i(e^t-1)}$$

Plugging this back we obtain

$$\mathbb{E}\left[e^{tX}\right] \leq \prod_{i=1}^n e^{p_i(e^t-1)} = e^{\mu(e^t-1)}$$

Putting back in (6.1), we obtain

$$\mathbb{P}[X \geq (1+\delta)\mu] \leq \frac{e^{\mu(e^t-1)}}{e^{t(1+\delta)\mu}} = e^{\mu(e^t-1-(1+\delta)t)} \underset{\text{set } t=\ln(1+\delta)}{=} \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$$

The other case can be proven similarly. □

6.1 Giant Connected Components in Erdös-Réyni Graphs

In this section we prove the following theorem.

Theorem 6.2. *Theorem 1 Let $\epsilon > 0$ be a small enough constant. Let G be an Erdös-Réyni random graph with parameter p.*

1. *Let $p = \frac{1-\epsilon}{n}$. Then whp all connected components of G are of size at most $\frac{7}{\epsilon^2}\ln n$.*
2. *Let $p = \frac{1+\epsilon}{n}$. Then whp G contains a path of length at least $\frac{\epsilon^2 n}{5}$.*

We run the DFS algorithm to prove the theorem. First, let us recall this algorithm: Fix a natural order $1 < 2 < \cdots < n$ on the vertices of G we assume that algorithm prioritizes vertices according to this natural order. DFS maintains three sets of vertices, letting X be the set of vertices whose exploration is complete, i.e., explored, U be the set of unvisited vertices, and $T = [n] \setminus X \setminus U$ be the set of active vertices in the stack.

The algorithm starts with $X = T = \varnothing$ and $U = V$, and runs till $T \cup U = \varnothing$. At each round of the algorithm, if the set T is non-empty, the algorithm queries U for neighbors of the last vertex v that has been added to T, scanning U according to the natural order. If v has a neighbor $u \in U$, the algorithm deletes u from U and inserts it into T . If v does not have a neighbor in U, then v is popped out of T and is moved to X. If T is empty, the algorithm chooses the first vertex of U according to the natural order, deletes it from U and pushes it into T. In order to complete the exploration of the graph, whenever the sets T and U have both become empty (at this stage all connected components of G have been revealed), we make the algorithm query all remaining pairs of vertices in S, not queried before.

The following properties of DFS are immediate:

- At each round of the algorithm one vertex moves, either from U to T , or from T to X;
- At any time during the algorithm, it has been revealed already that the graph G has no edges between the current set X and the current set of unvisited vertices U;
- The set T always spans a *path* (indeed, when a vertex u is added to T , it happens because u is a neighbor of the last vertex v in T ; thus, u augments the path spanned by T, of which v is the last vertex).

Let $N = \binom{n}{2}$ To prove the theorem we run DFS on a random input $G(n,p)$. Thus we feed DFS algorithm with a sequence of i.i.d. Bernoulli(p) random variables $Y_1, \dots, Y_N$ so that is gets its i-th query answered positively if $Y_i = 1$ and answered negatively otherwise, the so obtained graph is clearly distributed according to G(n, p). Thus, studying the component structure of G can be reduced to studying the properties of the random sequence X. In particular, observe crucially that as long as $U \neq \varnothing$, every positive answer to a query results in a vertex being moved from U to T , and thus after t queries and assuming $T \neq \varnothing$ still, we have $|X \cup T| \geq \sum_{i=1}^{t} Y_i$. (The last inequality is strict in fact as the first vertex of each connected component is moved from T to U "for free", i.e., without need to get a positive answer to a query.) On the other hand, since the addition of every vertex, but the first one in a connected component, to U is caused by a positive answer to a query, we have at time t: $|T| \leq 1 + \sum_{i=1}^{t} Y_i$.

The following lemma gives us the tool that we need to prove the theorem.

Lemma 6.3. *Let $\epsilon > 0$ be a small enough constant. Consider the sequence of iid Bernoulli random variables with parameter p. $Y_1, \dots, Y_N$.*

1. *Let $p = \frac{1-\epsilon}{n}$ and $k = \frac{7}{\epsilon^2}\ln n$. Then, with probability $\gtrsim 1 - 1/\sqrt{n}$, there is no interval of length kn where at least k of the Bernoullis are 1.*

2. Let $p = \frac{1+\epsilon}{n}$ and $N_0 = \frac{\epsilon n^2}{2}$. Then,

$$\mathbb{P}\left[\left|\sum_{i=1}^{N_0} Y_i - \frac{\epsilon(1+\epsilon)n}{2}\right| < n^{2/3}\right] \geq 1 - o(1).$$

Proof. Consider an interval I of length kn in $[N]$. Let $Y = \sum_{i \in I} Y_i$. Notice $\mathbb{E}[Y] = knp$. By the multiplicative Chernoff bound,

$$\mathbb{P}[Y \geq k] = \mathbb{P}\left[Y \geq \frac{\mathbb{E}[Y]}{np}\right] = \mathbb{P}\left[Y \geq \frac{\mathbb{E}[Y]}{1-\epsilon}\right] \leq \exp(-\frac{\epsilon^2 \mathbb{E}[Y]}{2+\epsilon}) \leq n^{-\frac{7}{2+\epsilon}}$$

where the last inequality follows by $k = \frac{7}{\epsilon^2} \ln n$. By a union bound the probability, since there are only $O(n^2)$ many such intervals the claim follows.

To prove 2, let $Y = \sum_{i=1}^{N_0} Y_i$. Then,

$$\mathbb{E}[Y] = N_0 \cdot p = \frac{(1+\epsilon)\epsilon n}{2}$$

Now, again by multiplicative Chernoff bound, for $\delta = \frac{2n^{-1/3}}{\epsilon(1+\epsilon)}$

$$\mathbb{P}\left[\left|\sum_{i=1}^{N_0} Y_i - \frac{\epsilon(1+\epsilon)n}{2}\right| > n^{2/3}\right] \leq \exp(-\delta^2\mu/3) \leq \exp(-n^{1/3})$$

□

We are now ready to prove the theorem.

Part 1. Assume to the contrary that G contains a connected component C with more than $k = \frac{7}{\epsilon^2} \ln n$ vertices. Let us look at the epoch of the DFS when C was created (an epoch is a period during which the stack gets empty again). Consider the moment inside this epoch when the algorithm has found the $(k+1)$-st vertex of C and is about to move it to T. Denote $X_C = X \cap C$ at that moment. Then $|X_C \cup T| = k$, and thus the algorithm got exactly k positive answers to its queries to random variables Y_i during the epoch, with each positive answer being responsible for revealing a new vertex of C, after the first vertex of C was put into T in the beginning of the epoch. During the epoch only pairs of edges touching $X_C \cup T$ have been queried, and the number of such pairs is therefore at most $\binom{k}{2} + k(n-k) \leq kn$. It thus follows that the sequence Y contains an interval of length at most kn with at least k 1's which is a contradiction.

Part 2. Now, assume that the sequence Y satisfies Property 2 of Lemma 6.3. We claim that after the first $N_0 = \frac{\epsilon n^2}{2}$ queries of the DFS algorithm, the set T contains at least $\frac{\epsilon^2 n}{5}$ vertices (with the contents of T forming a path of desired length at that moment).

First observe that $|X| < \frac{n}{3}$ at time N_0. Indeed, if $|X| \geq \frac{n}{3}$, then let us look at a moment t where $|X| = \frac{n}{3}$. At that moment $|T| \leq 1 + \sum_{i=1}^{t} Y_i < \frac{n}{3}$ by Property 2 of the Lemma. Then $|U| = n - |X| - |T| \geq \frac{n}{3}$, and the algorithm has examined all $|X| \cdot |U| \geq \frac{n^2}{9} > N_0$ pairs between X and U (and found them to be non-edges) – a contradiction.

Getting back to time N_0; now assume $|X| < \frac{n}{3}$ and $|T| < \frac{\epsilon^2}{5}$ then, we have $U \neq \varnothing$. This means in particular that the algorithm is still revealing connected components of G, and each positive answer it got resulted in moving a vertex from U to T (some of these vertices may have already moved further from T to X). By Property 2 of Lemma 6.3 the number of positive answers at that point is at least $\frac{\epsilon(1+\epsilon)n}{2} - n^{2/3}$. Hence, we have $|X \cup T| \geq \frac{\epsilon(1+\epsilon)n}{2} - n^{2/3}$. If $|T| \leq \frac{\epsilon^2 n}{5}$, then $|X| \geq \frac{\epsilon n}{2} + \frac{3\epsilon^2 n}{10} - n^{2/3}$. Therefore, all pairs of vertices between X, U are queried already (and received a negative answer), i.e., $|X| \cdot |U|$ many pairs. It follows that

$$\begin{aligned} \frac{\epsilon^2 n}{2} = N_0 &\geq |X| \cdot |U| \geq |X| \cdot \left(n - |X| - \frac{\epsilon^2 n}{5}\right) \\ &\geq \left(\frac{\epsilon n}{2} + \frac{3\epsilon^2 n}{10} - n^{2/3}\right) \cdot \left(n - \frac{\epsilon n}{2} - \frac{\epsilon^2 n}{2} + n^{2/3}\right) \\ &\geq \frac{\epsilon n^2}{2} + \frac{\epsilon^2 n^2}{20} - O(\epsilon^3) n^2 > \frac{\epsilon n^2}{2} \end{aligned}$$

as desired.

6.2 Congestion Minimization Problem

A classical technique in the field of approximation algorithms is to write down a linear programming relaxation of a combinatorial problem. The linear program (LP) is then solved in polynomial time, and one rounds the fractional solution to an integral solution that is, hopefully, not too much worse than the optimal solution.

A classical example goes back to Raghavan and Thompson [**RT87**]. Let $G = (V, A)$ be a directed network, and suppose that we are given a sequence of terminal pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ where $\{s_i\}, \{t_i\} \subseteq V$. The goal is to choose, for every i, a directed s_i-t_i path P_i in G so as to minimize the maximum congestion of an arc $e \in A$:

$$OPT = \min\{\max_{e \in A} \#\{i : e \in P_i\}\}$$

This problem is NP-hard. Our goal will be to design an approximation algorithm that outputs a solution so that the congestion of every edge is at most $\alpha \cdot OPT$, for α as small as possible. The number α is called the approximation factor of our algorithm. We will see that for this problem we will be able to achieve $\alpha = O(\frac{\log n}{\log \log n})$.

We start by writing a linear programming relaxation for this problem. Let $\mathcal{P}_i$ be the set of (directed) paths from s_i to t_i and let $\mathcal{P} = \cup_i \mathcal{P}_i$. For every path P, we have a variable x_P to denote the amount of flow that we route along P.

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & \sum_{P \in \mathcal{P}_i} y_P = 1 \qquad \forall 1 \leq i \leq k \\ & \sum_{P \in \mathcal{P} : e \in P} y_P \leq t \qquad \forall e \in A \\ & y_P \geq 0 \qquad \forall P \end{aligned} \tag{6.2}$$

A few observations are in order:

- $OPT(LP) \leq OPT$. This is simply because the optimum solution is a feasible solution in the above program. Note that the optimum solution satisfies all of the above constraints with the additional property that $y_P \in \{0, 1\}$ for all paths P.
- Although the above program has exponentially many variable one for every directed path connecting $s_i \to t_i$ (for all i), its optimum solution can be computed in polynomial time. To do that we need two observations:

i) We can write a linear program to find a flow of value 1 from s_i to t_i. We have a variable $f_e^{(i)}$ to denote the flow of every edge.

$$\begin{aligned}
&\sum_{s_i \to e} f_e^{(i)} = 1 \\
&\sum_{e \to v} f_e^{(i)} = \sum_{v \to e} f_e^{(i)} \quad && \forall v \neq s_i, t_i \\
&f_e^{(i)} \geq 0 && \forall e.
\end{aligned}$$

Having that, the congestion of e due to the flow routed between the i-th pair is $f_e^{(i)}$; so the total congestion of e is $\sum_i f_e^{(i)}$.

ii) A (fractional) flow (of value 1) from s_i to t_i can be decomposed into a distribution of paths from s_i to t_i. To see that, given the solution $\{f_e^{(i)}\}_{e \in A}$, greedily find a path P from s_i to t_i on edges with positive flow; let y_P be $\min_{e \in P} f_e^{(i)}$. Then, subtract y_P from the flow of all edges along P. We will obtain a new flow of value $1 - y_P$ from s_i to t_i. So we repeat this procedure until we get the 0 flow.

6.3 Independent Rounding

Given a solution y to (6.2), we want to round it to an integral solution. Namely, we want to choose exactly one path from each $\mathcal{P}_i$ such that the union of the chosen paths have small congestion, at most αOPT.

We follow the independent rounding method. Recall that, by feasiblity of y, for every $1 \leq i \leq k$, we know that $\sum_{P \in \mathcal{P}_i} y_P = 1$. So, we can think of $\{y_P\}_{P \in \mathcal{P}_i}$ as providing a probability distribution over s_i-t_i paths. For every i, independently, we choose one of the paths $P \in \mathcal{P}_i$ with probability y_P. This procedure, by definition, gives a feasible set of directed paths from s_i to t_i for all i. So, it remains to bound the maximum congestion. We prove the following theorem.

Theorem 6.4. *With probability at least $1-1/n$ the above algorithm produces a integral set of paths connecting all terminals with maximum congest at most*

$$C \frac{\log n}{\log \log n} OPT$$

Let Y_P be the indicator random variable that P is chosen. For an edge e, let X_e be the random variable that is the congestion of edge e. So,

$$X_e = \sum_{P \in \mathcal{P} : e \in P} Y_P.$$

By linearity of expectations,

$$\mathbb{E}[X_e] = \sum_{P \in \mathcal{P} : e \in P} y_P \leq OPT(LP).$$

So, the expectations are small. We just need to use a Chernoff bound/union bound argument. Unfortunately, the random variables Y_P are not independent. So, we need to use a slightly different random variables that are truly independent.

The idea is to note that we always have exactly one path from s_i to t_i. So, let $Y_{e,i}$ be the indicator random variable that the unique path from s_i to t_i uses edge e. So, $X_e = \sum_i Y_{e,i}$. It follows that $Y_{e,i}$'s are independent. Let $\beta = (1+\delta)\frac{OPT}{OPT(LP)}$, and note that $\beta > 1$ since $OPT \geq OPT(LP)$. So, by Chernoff bound,

$$\mathbb{P}\left[X_e \geq (1+\delta)OPT\right] = \mathbb{P}\left[X_e \geq \beta \cdot OPT(LP)\right] \leq \left(\frac{e^\beta}{\beta^\beta}\right)^{OPT(LP)} \leq \left(\frac{e^{1+\delta}}{(1+\delta)^{1+\delta}}\right)^{OPT} \leq \left(\frac{e^{1+\delta}}{(1+\delta)^{1+\delta}}\right)$$

where in the last inequality we simply use that $OPT \geq 1$. Now, to get the strong concentration bound we need to choose δ large enough such that the RHS is at most n^{-3}. It turns out that for that purpose it is enough to let $1+\delta = C\frac{\log n}{\log\log n}$ for a large enough constant $C > 1$.

Since, G has at most n^2 edges, $|A| \leq n^2$, by union bound

$$\mathbb{P}\left[\exists e : X_e \geq (1+\delta)OPT\right] \leq n^2 \cdot n^{-3} \leq \frac{1}{n}.$$

So, the algorithm succeeds with probability at least $1 - 1/n$.

6.4 Future Works and Open Problems

Chuzhoy, Guruswami, Khanna and Talwar showed that min-congestion problem is NP-hard to approximate within any factor better than $\frac{\log n}{\log\log n}$ when the underlying graph is directed. Note that the same algorithm that we discussed here works if the underlying graph is undirected (we can just put two copies of every edge one in every direction). However, for undirected, the result of Raghavan-Thompson is still the best known approximation factor. The best hardness result is $\log\log n$ by Andrews-Zhang. It is a fundamental open problem in the field of network routing to beat the Raghavan-Thompson's classical algorithm.