Mathematical Toolkit **Spring 2023**

Lecture 13: May 3, 2023

Lecturer: Avrim Blum

1 A small extension of Chernoff-Hoeffding bounds

One of the results we proved in the previous lecture is that if $X = X_1 + \ldots + X_n$ where the X_i are independent Bernoulli(p_i) random variables and $\mu = \mathbb{E}[X] = \sum_i p_i$, then for any $\delta \in [0,1]$ we have:

$$\mathbb{P}[X \geq (1+\delta)\mu] \quad \leq \quad e^{-\delta^2 \mu/3}.$$

In some cases (like the one we will discuss below) we will have an upper bound B on μ (i.e., $\mu \leq B$) and we would like to say that

$$\mathbb{P}[X \geq (1+\delta)B] \quad \leq \quad e^{-\delta^2 B/3}.$$

This indeed is legitimate, for the following reason. Define $p'_1, \ldots, p'_n \in [0,1]$ such that $p'_i \geq p_i$ for all i and $\sum_i p'_i = B$. (It is easy to see this is possible for any $B \in [\mu, n]$; if $B > n$ then the event in question cannot happen, so the desired inequality is trivially true). Now, define Bernoulli random variables $X'_1, \ldots, X'_n$ as follows: if $X_i = 1$ then $X'_i = 1$; else if $X_i = 0$ then $X'_i = 1$ with probability $\frac{p'_i - p_i}{1-p_i}$ and $X'_i = 0$ otherwise. It is not hard to see that X'_i is a Bernoulli(p'_i) random variable, and $X'_1, \ldots, X'_n$ are mutually independent. Therefore, defining $X' = X'_1 + \ldots + X'_n$, by Chernoff-Hoeffding bounds we have:

$$\mathbb{P}[X' \geq (1+\delta)B] \quad \leq \quad e^{-\delta^2 B/3}.$$

But, notice that $X' \geq X$ *always*. Therefore, we can replace X' with X in the above inequality as desired. Note that the same trick can be used for the general version of the Chernoff-Hoeffding bounds as well.

2 Low-congestion routing

Given a directed graph G and a set of pairs of vertices $\{(s_i, t_i)\}$, suppose we want to route these pairs to minimize the maximum congestion. That is, we want to find paths from each

s_i to its corresponding t_i such that no edge is used by more than C paths, for C as small as possible. This problem is NP-hard. Can we find an approximate solution?

Here is an idea: (by Raghavan & Thompson)

1. Solve the problem fractionally. Think of this as multi-commodity flow where we want to have one unit of flow of commodity i from s_i to t_i (e.g., allow s_i to route to t_i by sending 1/2 down one path, 1/4 down another path, and 1/4 down another). We can solve this with linear programming: for each (directed) edge (u, v), and each commodity i, we have a variable $x_{i,(u,v)}$. For each commodity i we have constraints that one unit of commodity i flows out of s_i, one unit of commodity i flows into t_i, and that the inflow of commodity i equals the outflow of commodity i for each vertex $v \notin \{s_i, t_i\}$. That is, $\sum_v x_{i,(s_i,v)} = 1$, $\sum_u x_{i,(u,t_i)} = 1$, and for all $v \notin \{s_i, t_i\}$ we have $\sum_u x_{i,(u,v)} = \sum_{u'} x_{i,(v,u')}$. Then for each edge (u, v) we add the constraint that $\sum_i x_{i,(u,v)} \leq C$, and minimize C.

2. Now, for each pair (s_i, t_i) we have a flow of one unit of commodity i. What we next do is view these fractional values $x_{i,(u,v)}$ as probabilities and select a path from s_i to t_i such that the probability we pick edge (u, v) is equal to the flow of this commodity on (u, v). How can we do this algorithmically? The claim is that a greedy approach will work: starting from s_i, examine all outgoing edges and select one (s_i, v_1) with probability proportional to the flow of commodity i on that edge; then repeat from the vertex v_1, and so on, continuing until t_i is reached.

Claim 2.1 *The greedy approach in step 2 above selects a path from s_i to t_i such that each edge (u, v) is selected with probability $x_{i,(u,v)}$.*

Proof: First, the flow of commodity i from s_i to t_i is a DAG (any cycles can be deleted), so we can sort the vertices of G that have flow of commodity i through them such that all edges (u, v) with $x_{i,(u,v)} > 0$ point forward in this ordering. Now, consider the vertices in this order. We will argue by induction.Base case: the first vertex is s_i itself, and all edges (s_i, v) are selected with probability equal to $x_{i,(s_i,v)}$ by construction. General case: consider now some arbitrary vertex $v \neq t_i$ in this ordering and assume inductively that each incoming edge (u, v) is selected with probability $x_{i,(u,v)}$. Since these events are disjoint (a path in a DAG cannot contain more than one edge into any given vertex), this means that v is reached with probability $= \sum_u x_{i,(u,v)}$. Conditioned on v being reached, each outgoing edge (v, w) is selected with probability $x_{i,(v,w)} / \sum_{u'} x_{i,(v,u')}$. But since $\sum_u x_{i,(u,v)} = \sum_{u'} x_{i,(v,u')}$, this implies the overall probability that (v, w) is selected (namely, the probability v is reached times the probability (v, w) is selected conditioned on v being reached) is exactly $x_{i,(v,w)}$ as desired. ■

We now use Claim 2.1 together with Chernoff-Hoeffding bounds and the union bound to prove that the solution found will with high probability be not too much worse than optimal. Specifically, define opt to be the optimal value for the congestion minimization problem. We then have:

Claim 2.2 *If* $\mathsf{opt} \gg \log n$ *(i.e.,* $\mathsf{opt} = \omega(\log n)$*) then with high probability this algorithm will find a solution with maximum congestion at most* $(1 + o(1))\mathsf{opt}$*. For any value of* opt*, this algorithm will with high probability find a solution that is within an* $O(\frac{\log n}{\log\log n})$ *factor of* opt*.*

Proof: Fix some edge (u, v) and let $X_{i,(u,v)}$ be the indicator random variable for the event that we picked (u, v) as an edge when selecting a path for routing commodity i. By Claim 2.1, we have $\mathbb{E}[X_{i,(u,v)}] = x_{i,(u,v)}$. Notice that while for a given commodity i, the associated random variables are highly dependent across edges, for a given edge (u, v) the associated random variables across commodities are completely independent. Moreover, if we define $X_{(u,v)} = \sum_i X_{i,(u,v)}$, we have $\mathbb{E}[X_{(u,v)}] \leq C$, where C is the quantity minimized in the linear program (the maximum congestion in the optimal fractional solution). Note also that since the linear program is a relaxation of the path-routing problem, we have $C \leq \mathsf{opt}$.

We can now apply Chernoff-Hoeffding bounds. Consider first the case that $\mathsf{opt} \gg \log n$. Using the extended version of the bounds from Section 1 and the fact that $\mathbb{E}[X_{(u,v)}] \leq \mathsf{opt}$, we have:

$$\mathbb{P}[X_{(u,v)} > (1 + \delta)\mathsf{opt}] \quad \leq \quad e^{-\delta^2 \mathsf{opt}/3}.$$

Since $\mathsf{opt} \gg \log n$, for any constant $\delta > 0$ the quantity on the right-hand-side is $o(1/n^2)$. So, by the union bound over all edges, the probability that *there exists* an edge whose congestion exceeds this bound is also small ($o(1)$).

What if $\mathsf{opt} = 1$, or opt is constant? In this case, we can apply the bound:

$$\mathbb{P}[X_{(u,v)} > k \ \mathsf{opt}] < \left(\frac{e^{k-1}}{k^k}\right)^{\mathsf{opt}} \leq \frac{e^{k-1}}{k^k}.$$

So, set k to be $\frac{3 \ln n}{\ln \ln n}$, and we again get a probability bound that is $o(1/n^2)$ as desired. ■

3 Randomized complexity classes

You have probably seen the class **P**, which refers to the class of problems that have polynomial-time deterministic algorithms. Here, we will look at randomized analogs of **P**. First, formally all of these classes refer to decision problems: problems whose answer is YES or NO. E.g., "Does the given graph have a perfect matching?" For such problems, we can split all possible instances into two categories: YES-instances (whose correct answer is YES) and

NO-instances (whose correct answer is NO). We can also put any ill-formed instances into the NO category.

Definition 3.1 *We say that an algorithm runs in* **Polynomial Time** *if, for some constant c, its running time is $O(n^c)$, where n is the size of the input.*

Definition 3.2 **P** *is the set of decision problems solvable by a deterministic polynomial-time algorithm.*

To define randomized complexity classes we will consider algorithms A that take in *two* inputs: an instance I of the problem being solved, and an auxiliary input y (a bit string) whose length is polynomial in the size of the instance I. Think of y as the random bits used by the algorithm. We can then define randomized complexity classes as follows:

Definition 3.3 *A problem Q is in* **RP** *if there exists a polynomial-time algorithm $A(I, y)$ and a polynomial r (the number of random bits requested) such that:*

- *If I is a YES-instance, then*

$$\mathop{\mathbb{P}}_{y \in \{0,1\}^{r(|I|)}}[A(I,y) = YES] \geq 1/2.$$

- *If I is a NO-instance, then*

$$\mathop{\mathbb{P}}_{y \in \{0,1\}^{r(|I|)}}[A(I,y) = YES] = 0.$$

The class **RP** corresponds to problems solvable by randomized algorithms with 1-sided error. For example, we showed that the perfect matching problem belongs to **RP** because we gave an algorithm such that if the given graph G has a perfect matching, then there is at least a $1/2$ chance our algorithm says YES (because the Tutte polynomial is nonzero), whereas if G has no perfect matching, then our algorithm is guaranteed to say NO (because the Tutte polynomial is identically zero). Note that the specific value "$1/2$" is arbitrary — any constant greater than 0 would give the same class (do you see why?).

There is also the complexity class **BPP** which corresponds to randomized algorithms with 2-sided error:

Definition 3.4 *A problem Q is in* **BPP** *if there exists a polynomial-time algorithm $A(I, y)$ and a polynomial r (the number of random bits requested) such that:*

- *If I is a YES-instance, then*

$$\mathop{\mathbb{P}}_{y \in \{0,1\}^{r(|I|)}}[A(I,y) = YES] \geq 3/4.$$

- *If I is a NO-instance, then*

$$\mathop{\mathbb{P}}_{y \in \{0,1\}^{r(|I|)}}[A(I,y) = YES] \leq 1/4.$$

Again, the specific constants $3/4$ and $1/4$ are arbitrary: any two constants with one larger than the other would give the same class (can you see why?).

It is believed that $\mathbf{P} = \mathbf{BPP}$, though there is no deterministic polynomial-time algorithm known for the polynomial identity testing problem.

One more complexity class to mention is **P/poly**. This is the class of problems solvable in "non-uniform polynomial time".

Definition 3.5 *A problem Q is in* **P/poly** *if there exists a polynomial-time algorithm $A(I, y)$ and a polynomial r such that for every $n = |I|$ there exists a string $y_n \in \{0,1\}^{r(n)}$ such that $A(I, y_{|I|})$ is always correct.*

You can think of y_n is an "advice" string used for instances of size n. Note that because y_n has length polynomial in n, it can't just be a list of the answers for all instances I of size n (because there could be up to 2^n such instances). One interesting fact is that $\mathbf{RP} \subseteq \mathbf{P/poly}$:

Theorem 3.6 $\mathbf{RP} \subseteq \mathbf{P/poly}$.

Proof: Suppose $Q \in \mathbf{RP}$. Then there exists an algorithm A and polynomial r satisfying the conditions of the **RP** definition. Consider an algorithm A' that given an instance I of size n uses an auxiliary random input y_n of length $(n+1)r(n)$ to peform $n+1$ runs of A, outputting YES if any of the runs outputs YES and outputting NO otherwise. The probability that A' fails on a given instance I of size n is at most $1/2^{n+1}$. Since there are at most 2^n instances I of size n, by the union bound, the chance for a random y_n that there *exists* an instance I of size n for which $A'(I, y_n)$ fails is at most $2^n/2^{n+1} = 1/2$. Therefore, a string y_n causing A' to succeed on all instances of size n must exist. ■