Lecture 4

Prof. Nick Harvey *University of British Columbia*

We begin today's lecture by resuming our discussion of the congestion minimization problem. Next, we will discuss the **negative binomial distribution**, another distribution that often arises in analyzing randomized algorithms. We conclude by using that distribution to analyze the randomized QuickSort algorithm.

1 Congestion Minimization, Continued

Recall that last time we were discussing the congestion minimization problem, which we may define as the following integer program:

$$\begin{array}{lll} \min & C & \\ \text{s.t.} & \sum_{P \in \mathcal{P}_i} x_P^i = 1 & \forall i = 1, \ldots, k \\ & \sum_i \sum_{P \in \mathcal{P}_i \text{ with } a \in P} x_P^i \leq C & \forall a \in A \\ & x_P^i \in \{0,1\} & \forall i = 1, \ldots, k \text{ and } P \in \mathcal{P}_i \end{array}$$

Here, the set $\mathcal{P}_i$ consists of all paths between the nodes s_i and t_i.

That problem is NP-hard to solve, so we "relax" it into a linear program, basically by replacing the integrality constraints with non-negativity constraints. It turns out to be convenient also to add the constraint $C \geq 1$. The resulting linear program is:

$$\begin{array}{lll} \min & C & \\ \text{s.t.} & \sum_{P \in \mathcal{P}_i} x_P^i = 1 & \forall i = 1, \ldots, k \\ & \sum_i \sum_{P \in \mathcal{P}_i \text{ with } a \in P} x_P^i \leq C & \forall a \in A \\ & C \geq 1 & \\ & x_P^i \geq 0 & \forall i = 1, \ldots, k \text{ and } P \in \mathcal{P}_i \end{array}$$

Remarkably, this LP can be solved in time polynomial in n (the number of nodes of G), even though its number of variables could be exponential in n. The details are best left for a course on optimization[1]. Our algorithm will solve this LP and obtain a solution where the number of non-zero x_P^i variables is only polynomial in n. Let C^* be the optimum value of the LP.

Claim 1 $C^* \leq OPT$.

Proof: The LP was obtained from the IP by *removing* constraints. Therefore any feasible solution for the IP is also feasible for the LP. In particular, the optimal solution for the IP is feasible for the LP. So the LP has a solution with objective value equal to OPT. □

[1] There are two ways to do this. The first way is to solve the dual LP using the ellipsoid method. This can be done in poly(n) time even though it can have exponentially many constraints. The second way is to find a "compact formulation" of the LP which uses fewer variables, much like the usual LP that you may have seen for the ordinary maximum flow problem.

The Rounding. Our algorithm will solve the LP and most likely obtain a "fractional" solution — a solution with some non-integral variables, which is therefore not feasible for the IP. The next step of the algorithm is to "round" that fractional solution into a solution which is is feasible for the IP. In doing so, the congestion might increase, but we will ensure that it does not increase too much.

The technique we will use is called **randomized rounding**. For each each $i = 1, \ldots, k$, we randomly choose *exactly one path* P_i by setting $P_i = P$ with probability x_P^i. (The LP's constraints ensure that these are indeed probabilities: they are non-negative and sum up to 1.) The algorithm outputs the chosen paths $P_1, \ldots, P_k$.

Analysis. All that remains is to analyze the congestion of these paths. Let Y_i^a be the indicator random variable that is 1 if $a \in P_i$ and 0 otherwise. Let $Y^a = \sum_i Y_i^a$ be the congestion on arc a. The expected value of Y^a is easy to analyze:

$$\mathrm{E}[\,Y^a\,] \;=\; \sum_i \mathrm{E}[\,Y_i^a\,] \;=\; \sum_i \sum_{P \in \mathcal{P}_i \text{ with } a \in P} x_P^i \;\le\; C^*,$$

where the inequality comes from the LP's second constraint. (Recall we assume that the fractional solution is optimal for the LP, and therefore $C = C^*$.)

The Chernoff bound says, if X is a sum of independent random variables each of which take values in $[0, 1]$, and μ is an *upper bound* on $\mathrm{E}[\,X\,]$, then

$$\Pr[\,X \ge (1+\delta)\mu\,] \;\le\; \exp\Big(-\mu \cdot \big((1+\delta)\ln(1+\delta) - \delta\big)\Big) \qquad \forall \delta > 0.$$

We apply this to Y^a, taking $\mu = C^*$ and $\alpha = 1 + \delta = 6 \log n / \log\log n$. Following our balls-and-bins argument from last time,

$$\begin{aligned} \Pr[\,Y^a \ge \alpha C^*\,] \quad &\le \quad \exp\Big(-C^*\big(\alpha \ln \alpha - (\alpha - 1)\big)\Big) \\ &\le \quad \exp\big(-\alpha \ln \alpha + \alpha - 1\big) \\ &\le \quad \exp\big(-(6/2)\ln n\big) \;=\; 1/n^3. \end{aligned}$$

We now use a union bound to analyze the probability of *any* arc having congestion greater than αC^*.

$$\Pr[\,\text{any } a \text{ has } Y^a \ge \alpha C^*\,] \;\le\; \sum_{a \in A} \Pr[\,Y^a \ge \alpha C^*\,] \;\le\; \sum_{a \in A} 1/n^3 \;\le\; 1/n,$$

since the graph has at most n^2 arcs. So, with probability at least $1 - 1/n$, the algorithm produces a solution for which every arc has congestion at most αC^*, which is at most $\alpha \cdot OPT$ by Claim 1. So our algorithm has approximation factor $\alpha = O(\log n / \log\log n)$.

Further Remarks. The *rounding* algorithm that we presented is actually optimal: there are graphs for which $OPT/C^* = \Omega(\log n / \log\log n)$. Consequently, every rounding algorithm which converts a fractional solution of LP to an integral solution of IP must necessarily incur an increase of $\Omega(\log n / \log\log n)$ in the congestion.

That statement does not rule out the possibility that there is a better algorithm which behaves completely differently (i.e., one which does not use IP or LP at all). But sadly it turns out that there is no better algorithm (for the case of directed graphs). It is known that every efficient algorithm must have approximation factor $\alpha = \Omega(\log n / \log\log n)$, assuming a reasonable complexity theoretic conjecture ($\mathrm{NP} \not\subseteq \mathrm{BPTIME}(n^{O(\log\log n)})$). So the algorithm that we presented is optimal, up to constant factors.

2 The Negative Binomial Distribution

The **negative binomial distribution** is a woefully underappreciated distribution. It shows up in many different randomized algorithms, but it is not taught or covered in textbooks as much as it should be.

There are a few ways to define this distribution. We adopt the following definition. There are two parameters, $p \in [0,1]$ and $k \in \mathbb{N}$. Suppose we perform a sequence of independent Bernoulli trials, each succeeding with probability p. Let Y be the number of trials performed until we see the kth success. Then Y is said to have the negative binomial distribution.

Note that this is quite different from the usual binomial distribution. For example, if X is a binomial random variable with parameters n and p then the value of X is always at most n. In contrast, Y has positive probability of taking any integer value larger than or equal to k. Nevertheless, there is a relationship between the tails of X and Y. The following claim is quite useful, although hard to find in the literature.

Claim 2 *Let $n, k \in \mathbb{N}$ and $p \in (0,1]$. Let Y be a random variable distributed according to the negative binomial distribution with parameters k and p. Let X be a random variable distributed according to the binomial distribution with parameters n and p. Then $\Pr[Y > n] = \Pr[X < k]$.*

Informally, this is quite easy to see. Roughly, both events say that "after performing n trials, we still have not seen k successes". That argument is not completely formal because the sample spaces of X and Y are not the same.[2]

An important consequence of the previous claim is that Chernoff bounds give tail bounds on Y.

Claim 3 *Let Y have the negative binomial distribution with parameters k and p. Pick $\delta \in (0,1)$ and set $n = \left\lceil \frac{k}{(1-\delta)p} \right\rceil$. Then $\Pr[Y > n] \leq \exp\big(-\delta^2 k/2(1-\delta)\big)$.*

Proof: Let X have the binomial distribution with parameters n and p. Note that $\mathrm{E}[X] = np \geq k/(1-\delta)$; this last quantity will be denoted by $\mu_{\min}$. Then

$$\begin{aligned}
\Pr[Y > n] &= \Pr[X < k] \qquad \text{(by Claim 2)} \\
&= \Pr[X < (1-\delta)\mu_{\min}] \\
&\leq \exp(-\delta^2 \mu_{\min}/2) \qquad \text{(by Chernoff bound)} \\
&= \exp\big(-\delta^2 k/2(1-\delta)\big).
\end{aligned}$$

□

3 Example: Quicksort

Quicksort is one of the most famous algorithms for sorting an array of comparable elements. (We assume for simplicity that the array has no equal elements.) The quicksort algorithm is recursive. In each recursive call it picks a pivot, then partitions the current array into two parts: the elements that

[2] The easiest way to make this formal is using a method known as "coupling" to put X and Y into the same probability space. Another way is to explicitly compute the probabilities $\Pr[Y > n]$ and $\Pr[X < k]$, then show that they are equal using identities involving binomial coefficients.

are strictly smaller than the pivot, and the elements that are at least the pivot. This partitioning process takes time that is linear in the size of the current array. It then recursively sorts the two parts, stopping the recursion when the current array consists of a single element.

It is well-known that quicksort probably takes $O(m \log m)$ time to sort an array of length m if each partitioning step chooses the pivot element uniformly at random from the current array. There are many ways to prove this fact. We now give a short proof using Chernoff bounds applied to random variables with the negative binomial distribution.

Let m be the size of the original input array. Notice that the total amount of work done by all subroutines at level i of the recursion is $O(m)$, since each element of the input array appears in at most one subproblem at level i. So we obtain a runtime bound of $O(mL)$ if we can show that the maximum level of any recursive subproblem is L.

Every leaf of the recursion corresponds to a distinct element of the input array, so there are exactly m leaves. We will show that every leaf has level $O(\log m)$. To do so, fix a particular element x of the input array and consider all partitioning steps involving x. Intuitively, we would like each recursive call to partition the current array into two halves of nearly equal size. To formalize this, we say that a partitioning step is "good" if it partitions the current array into two parts, both of which have size *at least one third* of that array. So the probability of being good is $1/3$.

Each good partitioning step shrinks the size of the current array by a factor of $2/3$ or better, so after $\log_{3/2}(m) < 3 \ln m$ good partitioning steps the current array has size at most 1. So x can be involved in at most $3 \ln m$ good partitioning steps, irrespective of the random decisions. Our only worry is that x could be involved in many bad partitioning steps. We can upper bound the number of partitioning steps involving x using a negative binomially distributed random variable: the number of trials needed to see $k = 3 \ln m$ successes when the probability of success is $1/3$.

We use Claim 3 with $\delta = 2/3$, so $n = \lceil 27 \ln m \rceil$. Then the probability that more than n trials are needed to see k successes is at most $\exp(-\delta^2(3 \ln m)/2(1-\delta)) = m^{-2}$. So the number of partitioning steps involving x (i.e., the depth of the leaf containing x) exceeds n with probability at most m^{-2}. Taking a union bound over all m elements of the array, the probability that any leaf has depth greater than n is at most $1/m$. Therefore the running time is $O(mn) = O(m \log m)$ with probability at least $1 - 1/m$.

References: Mitzenmacher-Upfal Exercise 4.20, Dubhashi-Panconesi Section 2.4.