CS261: A Second Course in Algorithms
Lecture #18: Five Essential Tools for the Analysis of Randomized Algorithms*

Tim Roughgarden†

March 3, 2016

1 Preamble

In CS109 and CS161, you learned some tricks of the trade in the analysis of randomized algorithms, with applications to the analysis of QuickSort and hashing. There's also CS265, where you'll learn more than you ever wanted to know about randomized algorithms (but a great class, you should take it). In CS261, we build a bridge between what's covered in CS161 and CS265. Specifically, this lecture covers five essential tools for the analysis of randomized algorithms. Some you've probably seen before (like linearity of expectation and the union bound) while others may be new (like Chernoff bounds). You will need these tools in most 200- and 300-level theory courses that you may take in the future, and in other courses (like in machine learning) as well. We'll point out some applications in approximation algorithms, but keep in mind that these tools are used constantly across all of theoretical computer science.

Recall the standard probability setup. There is a *state space* Ω; for our purposes, Ω is always finite, for example corresponding to the coin flip outcomes of a randomized algorithm. A *random variable* is a real-valued function $X : \Omega \to \mathbb{R}$ defined on Ω. For example, for a fixed instance of a problem, we might be interested in the running time or solution quality produced by a randomized algorithm (as a function of the algorithm's coin flips). The *expectation* of a random variable is just its average value, with the averaging weights given by a specified probability distribution on Ω:

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} \mathbf{Pr}[\omega] \cdot X(\omega).$$

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

An *event* is a subset of Ω. The *indicator random variable* for an event $E \subseteq \Omega$ takes on the value 1 for $\omega \in E$ and 0 for $\omega \notin E$. Two events E_1, E_2 are *independent* if their probabilities factor: $\mathbf{Pr}[E_1 \wedge E_2] = \mathbf{Pr}[E_1] \cdot \mathbf{Pr}[E_2]$. Two random variables X_1, X_2 are independent if, for every x_1 and x_2, the events $\{\omega : X_1(\omega) = x_1\}$ and $\{\omega : X_2(\omega) = x_2\}$ are independent. In this case, expectations factor: $\mathbf{E}[XY] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$. Independence for sets of 3 or more events or random variables is defined analogously (for every subset, probabilities should factor). Probabilities and expectations generally don't factor for non-independent random variables, for example if E_1, E_2 are complementary events (so $\mathbf{Pr}[E_1 \wedge E_2] = 0$).

2 Linearity of Expectation and MAX 3SAT

2.1 Linearity of Expectation

The first of our five essential tools is *linearity of expectation*. Like most of these tools, it somehow manages to be both near-trivial and insanely useful. You've surely seen it before.[1] To remind you, suppose $X_1, \ldots, X_n$ are random variables defined on a common state space Ω. Crucially, *the X_i's need not be independent*. Linearity of expectation says that we can freely exchange expectations with summations:

$$\mathbf{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbf{E}[X_i]\,.$$

The proof is trivial — just expand the expectations as sums over Ω, and reverse the order of summation.

The analogous statement for, say, products of random variables is not generally true (when the X_i's are not independent). Again, just think of two indicator random variables for complementary events.

As an algorithm designer, why should you care about linearity of expectation? A typical use case works as follows. Suppose there is some complex random variable X that we care about — like the number of comparisons used by QuickSort, or the objective function value of the solution returned by some randomized algorithm. In many cases, it is possible to express the complex random variable X as the sum $\sum_{i=1}^{n} X_i$ of much simpler random variables $X_1, \ldots, X_n$, for example indicator random variables. One can then analyze the expectation of the simple random variables directly, and exploit linearity of expectation to deduce the expected value of the complex random variable of interest. You should have seen this recipe in action already in CS109 and/or CS161, for example when analyzing QuickSort or hash tables. Remarkably, linearity of expectation is already enough to derive interesting results in approximation algorithms.

[1] When I teach CS161, out of all the twenty lectures, exactly one equation gets a box drawn around it for emphasis — linearity of expectation.

2.2 A $\frac{7}{8}$-Approximation Algorithm for MAX 3SAT

An input of *MAX 3SAT* is just like an input of 3SAT — there are n Boolean variables $x_1, \ldots, x_n$ and m clauses. Each clause is the disjunction ("or") of 3 literals (where a literal is a variable or its negation). For example, a clause might have the form $x_3 \vee \neg x_6 \vee \neg x_{10}$. For simplicity, assume that the 3 literals in each clause correspond to distinct variables. The goal is to output a truth assignment (an assignment of each x_i to { true, false }) that satisfies the maximum-possible number of clauses. Since 3SAT is the special case of checking whether or not the optimal objective function value equals m, MAX 3SAT is an NP-hard problem.

A very simple algorithm has a pretty good approximation ratio.

Theorem 2.1 *The expected number of clauses satisfied by a random truth assignment, chosen uniformly at random from all 2^n truth assignments, is $\frac{7}{8}m$.*

Since the optimal solution can't possibly satisfy more than m clauses, we conclude that the algorithm that chooses a random assignment is a $\frac{7}{8}$-approximation (in expectation).

Proof of Theorem 2.1: Identify the state space Ω with all 2^n possible truth assignments (with the uniform distribution). For each clause j, let X_j denote the indicator random variable for the event that clause j is satisfied. Observe that the random variable X that we really care about, the number of satisfied clauses, is the sum $\sum_{j=1}^{n} X_j$ of these simple random variables. We now follow the recipe above, analyzing the simple random variables directly and using linearity of expectation to analyze X. As always with an indicator random variable, the expectation is just the probability of the corresponding event:

$$\mathbf{E}[X_j] = 1 \cdot \mathbf{Pr}[X_j = 1] + 0 \cdot \mathbf{Pr}[X_j = 0] = \mathbf{Pr}[\text{clause } j \text{ satisfied}]\,.$$

The key observation is that clause j is satisfied by a random assignment with probability exactly $\frac{7}{8}$. For example, suppose the clause is $x_1 \vee x_2 \vee x_3$. Then a random truth assignment satisfies the clause unless we are unlucky enough to set each of x_1, x_2, x_3 to false — for all of the other 7 combinations, at least one variable is true and hence the clause is satisfied. But there's nothing special about this clause — for any clause with 3 literals corresponding to distinct variables, only 1 of the 8 possible assignments to these three variables fails to satisfy the clause.

Putting the pieces together and using linearity of expectation, we have

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{j=1}^{m} X_j\right] = \sum_{j=1}^{m} \mathbf{E}[X_j] = \sum_{j=1}^{m} \tfrac{7}{8} = \tfrac{7}{8}m,$$

as claimed. ■

If a random assignment satisfies $\frac{7}{8}m$ clauses on average, then certainly some truth assignment does as well as this average.[2]

[2]It is not hard to derandomize the randomized algorithm to compute such a truth assignment deterministically in polynomial time, but this is outside the scope of this lecture.

Corollary 2.2 *For every 3SAT formula, there exists a truth assignment satisfying at least 87.5% of the clauses.*

Corollary 2.2 is counterintuitive to many people the first time they see it, but it is a near-trivial consequence of linearity of expectation (which itself is near-trivial!).

Remarkably, and perhaps depressingly, there is no better approximation algorithm: assuming $P \neq NP$, there is no $(\frac{7}{8} + \epsilon)$-approximation algorithm for MAX 3SAT, for any constant $\epsilon > 0$. This is one of the major results in "hardness of approximation."

3 Tail Inequalities

If you only care about the expected value of a random variable, then linearity of expectation is often the only tool you need. But in many cases one wants to prove that an algorithm is good not only on average, but is also good almost all the time ("with high probability"). Such high-probability statements require different tools.

The point of a *tail inequality* is to prove that a random variable is very likely to be close to its expected value — that the random variable "concentrates." In the world of tail inequalities, there is always a trade-off between how much you assume about your random variable, and the degree of concentration that you can prove. This section looks at the three most commonly used points on this trade-off curve. We use hashing as a simple running example to illustrate these three inequalities; the next section connects these ideas back to approximation algorithms.

3.1 Hashing

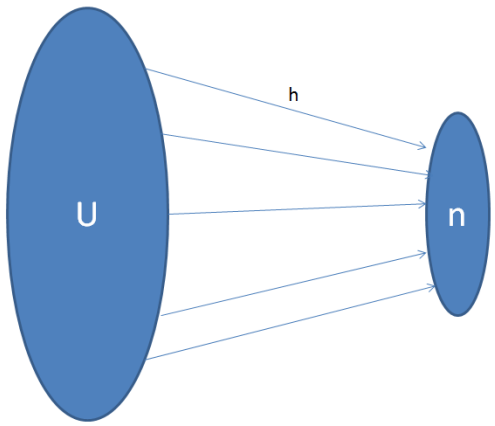

Figure 1: a hash function h that maps a large universe U to a relatively smaller number of buckets n.

Throughout this section, we consider a family $\mathcal{H}$ of hash functions, with each $h \in \mathcal{H}$ mapping a large universe U to a relatively small number of "buckets" $\{1, 2, \ldots, n\}$ (Figure 1). We'll be thinking about the following experiment, which should be familiar from CS161: an adversary picks an arbitrary data set $S \subseteq U$, then we pick a hash function $h \in \mathcal{H}$ uniformly at random and use it to hash all of the elements of S. We'd like these objects to be distributed evenly across the buckets, and the maximum load of a bucket (i.e., the number of items hashing to it) is a natural measure of distance from this ideal case. For example, in a hash table with chaining, the maximum load of a bucket governs the worst-case search time, a highly relevant statistic.

3.2 Markov's Inequality

For now, all we assume about $\mathcal{H}$ is that each object is equally likely to map to each bucket (though not necessarily independently).

(P1) For every $x \in U$ and $i \in \{1, 2, \ldots, n\}$, $\mathbf{Pr}_{h \in \mathcal{H}}[h(x) = i] = \frac{1}{n}$.

This property is already enough to analyze the expected load of a bucket. For simplicity, suppose that the size $|S|$ of the data set being hashed equals the number of buckets n. Then, for any bucket i, by linearity of expectation (applied to indicator random variables for elements getting mapped to i), its expected load is

$$\sum_{x \in S} \underbrace{\mathbf{Pr}[h(x) = i]}_{=1/n \text{ by (P1)}} = \frac{|S|}{n} = 1. \tag{1}$$

This is good — the expectations seem to indicate that things are balanced on average. But can we prove a concentration result, stating that loads are close to these expectations?

The following tail inequality gives a weak bound but applies under minimal assumptions; it is our second (of 5) essential tools for the analysis of randomized algorithms.

Theorem 3.1 (Markov's Inequality) *If X is a non-negative random variable with finite expectation, then for every constant $c \geq 1$,*

$$\mathbf{Pr}[X \geq c \cdot \mathbf{E}[X]] \leq \frac{1}{c}.$$

For example, such a random variable is at least 10 times its expectation at most 10% of the time, and is at least 100 times its expectation at most 1% of the time. In general, Markov's inequality is useful when a constant probability guarantee is good enough. The proof of Markov's inequality is easy, and we leave it to Exercise Set #9.[3]

[3]Both hypotheses are necessary. For example, random variables that are equally likely to be M or $-M$ exhibit no concentration whatsoever as $M \to \infty$.

We now apply Markov's inequality to the random variable equal to the load of our favorite bucket i. We can choose any $c \geq 1$ we want in Theorem 3.1. For example, choosing $c = n$ and recalling that the relevant expectation is 1 (assuming $|S| = n$), we obtain

$$\mathbf{Pr}[\text{load of i } \geq n] \leq \frac{1}{n}.$$

The good news is that $\frac{1}{n}$ is not a very big number when n is large. But let's look at the event we're talking about: the load of i being at least n means that *every single element* of S hashes to i. And this sounds crazy, like it should happen much less often than $1/n$ of the time. (If you hash 100 things into a hash table with 100 buckets, would you really expect everything to hash to the same bucket 1% of the time?)

If we're only assuming the property (P1), however, it's impossible to prove a better bound. To see this, consider the set $\mathcal{H} = \{h(x) = i \,:\, i = 1, 2, \ldots, n\}$ of constant hash functions, each of which maps all items to the same bucket. Observe that $\mathcal{H}$ satisfies property (P1). But the probability that all items hash to the bucket i is indeed $\frac{1}{n}$.

3.3 Chebyshev's Inequality

A totally reasonable objection is that the example above is a stupid family of hash function that no one would ever use. So what about a good family of hash functions, like those you studied in CS161? Specifically, we now assume:

(P2) for every pair $x, y \in U$ of distinct elements, and every $i, j \in \{1, 2, \ldots, n\}$,

$$\mathbf{Pr}_{h \in \mathcal{H}}[h(x) = i \text{ and } h(y) = j] = \frac{1}{n^2}.$$

That is, when looking at only two elements, the joint distribution of their buckets is as if the function h is a totally random function. (Property (P1) asserts an analogous statement when looking at only a single element.) A family of hash functions satisfying (P2) is called a *pairwise* or *2-wise* independent family. This is almost the same as (and for practical purposes equivalent to) the notion of "universal hashing" that you saw in CS161. The family of constant hash functions (above) clearly fails to satisfy property (P2).

So how do we use this stronger assumption to prove sharper concentration bounds? Recall that the *variance* $\text{Var}[X]$ of a random variable is its expected squared deviation from its mean $\mathbf{E}[(X - \mathbf{E}[X])^2]$, and that the *standard deviation* is the square root of the variance. Assumption (P2) buys us control over the variance of the load of a bucket. *Chebyshev's inequality*, the third of our five essential tools, is the inequality you want to use when the best thing you've got going for you is a good bound on the variance of a random variable.

Theorem 3.2 (Chebyshev's Inequality) *If X is a random variable with finite expectation and variance, then for every constant $t \geq 1$,*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| > t \cdot \mathit{StdDev}[X]] \leq \frac{1}{t^2}.$$

For example, the probability that a random variable differs from its expectation by at least two standard deviations is at most 25%, and the probability that it differs by at least 10 standard deviations is at most 1%. Chebyshev's inequality follows easily from Markov's inequality; see Exercise Set #9.

Now let's go back to the load of our favorite bucket i, where a data set $S \subseteq U$ with size $|S| = n$ is hashed using a hash function h chosen uniformly at random from $\mathcal{H}$. Call this random variable X. We can write

$$X = \sum_{y \in S} X_y,$$

where X_y is the indicator random variable for whether or not $h(y) = i$. We noted earlier that, by (P1), $\mathbf{E}[X] = \sum_{y \in S} \frac{1}{n} = 1$.

Now consider the variance of X. We claim that

$$\text{Var}[X] = \sum_{y \in S} \text{Var}[X_y], \tag{2}$$

analogous to linearity of expectation. Note that this statement is *not* true in general — e.g., if X_1 and X_2 are indicator random variables of complementary events, then $X_1 + X_2$ is always equal to 1 and hence has variance 0. In CS109 you saw a proof that for independent random variables, variances add as in (2). If you go back and look at this derivation — seriously, go look at it — you'll see that the variance of a sum equals the sum of the variances of the summands, plus correction terms that involve the covariances of pairs of summands. The covariance of independent random variables is zero. Here, we are only dealing with pairwise independent random variables (by assumption (P2)), but still, this implies that the covariance of any two summands is 0. We conclude that (2) holds not only for sums of independent random variables, but also of pairwise independent random variables.

Each indicator random variable X_y is a Bernoulli variable with parameter $\frac{1}{n}$, and so $\text{Var}[X_y] = \frac{1}{n}(1 - \frac{1}{n}) \leq 1$. Using (2), we have $\text{Var}[X] = \sum_{y \in S} \text{Var}[X_y] \leq n \cdot \frac{1}{n} = 1$. (By contrast, when $\mathcal{H}$ is the set of constant hash functions, $\text{Var}[X] \approx n$.)

Applying Chebyshev's inequality with $t = n$ (and ignoring "+1" terms for simplicity), we obtain

$$\mathbf{Pr}_{h \in \mathcal{H}}[X \geq n] \leq \frac{1}{n^2}.$$

This is a better bound than what we got from Markov's inequality, but it still doesn't seem that small — when hashing 10 elements into 10 buckets, do you really expect to see all of them in a single bucket 1% of the time? But again, without assuming more than property (P2), we can't do better — there exist families of pairwise independent hash functions such that all elements hash to the same bucket with probability $\frac{1}{n^2}$; showing this is a nice puzzle.

3.4 Chernoff Bounds

In this section we assume that:

(P3) all $h(x)$'s are uniformly and independently distributed in $\{1, 2, \ldots, n\}$. Equivalently, h is completely random function.

How can we use this strong assumption to prove sharper concentration bounds?

The fourth of our five essential tools for analyzing randomized algorithms is the *Chernoff bounds*. They are the centerpiece of this lecture, and are used all the time in the analysis of algorithms (and also complexity theory, machine learning, etc.).

The point of the Chernoff bounds is to prove sharp concentration for sums of independent and bounded random variables.

Theorem 3.3 (Chernoff Bounds) *Let $X_1, \ldots, X_n$ be random variables, defined on the same state space and taking values in $[0,1]$, and set $X = \sum_{j=1}^{n} X_j$. Then:*

(i) for every $\delta > 0$,

$$\mathbf{Pr}[X > (1+\delta)\mathbf{E}[X]] < \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mathbf{E}[X]}.$$

(ii) for every $\delta \in (0,1)$,

$$\mathbf{Pr}[X < (1-\delta)\mathbf{E}[X]] < e^{-\delta^2\mathbf{E}[X]/2}.$$

The key thing to notice in Theorem 3.3 is that the deviation probability decays exponentially in both the factor of the deviation $(1+\delta)$ and the expectation of the random variable ($\mathbf{E}[X]$). So if either of these quantities is even modestly big, then the deviation probability is going to be very small.[4]

We could prove Theorem 3.3 in 30 minutes or less, but the right place to spend time on the proof is a randomized algorithms class (like CS265). So we'll just use the Chernoff bounds as a "black box" — this is how almost everybody thinks about them, anyways. It's notable that, of our five essential tools for the analysis of randomized algorithms, only the Chernoff bounds require a non-trivial proof. We'll only use part (i) in this lecture, but (ii) is also useful in many situations. An analog of Theorem 3.3 for random variables that are nonnegative and bounded (not necessarily in $[0,1]$) follows from a simple scaling argument. The independence assumption can be relaxed, for example to negatively correlated random variables, although the proof then requires a bit more work.

Now let's apply the Chernoff bounds to analyze the number of items hashing to our favorite bucket i, under the assumption (P3) that h is a uniformly random function. Again using X_y to denote the indicator random variable for the event that $h(y) = i$, we see that $X = \sum_{y \in S} X_y$ is now the sum of independent 0-1 random variables, and hence is right in the wheelhouse of the Chernoff bounds. For example, setting $1 + \delta = \ln n$ and recalling that $\mathbf{E}[X] = 1$, Theorem 3.3 implies that

$$\mathbf{Pr}[X > \ln n] < \left(\frac{e}{\ln n}\right)^{\ln n}. \tag{3}$$

To interpret this bound, note that $(\frac{1}{e})^{\ln n} = \frac{1}{n}$. More generally, a constant less than one raised to a logarithmic power yields an inverse polynomial. Now $\frac{e}{\ln n}$ is smaller than any

[4] For the first bound (i), it is common to state the tighter probability upper bound of $[e^{\delta}/(1+\delta)^{(1+\delta)}]^{\mathbf{E}[X]}$, which is useful in applications where δ is small. The simpler bound here suffices for all of our applications.

constant as n grows large, and hence the probability bound in (3) is smaller than any inverse polynomial. Notice how much better this is than what we could prove using Markov's or Chebyshev's inequality — we're looking at a much smaller deviation ($\ln n$ instead of n) yet obtaining a much smaller probability bound (smaller than any inverse polynomial).

Theorem 3.3 even implies that

$$\mathbf{Pr}\left[X > \frac{3\ln n}{\ln\ln n}\right] \leq \frac{1}{n^2}, \tag{4}$$

as you should verify. Why $\ln n/\ln\ln n$? Because this is roughly the solution to the equation $x^x = n$ (this is relevant in Theorem 3.3 because of the $(1+\delta)^{-(1+\delta)}$ term). Again, this is a huge improvement over what we obtained using Markov's and Chebyshev's inequalities. For a more direct comparison, note that Chernoff bounds imply that the probability $\mathbf{Pr}[X \geq n]$ is at most an inverse exponential function of n (as opposed to an inverse polynomial function).

3.5 The Union Bound

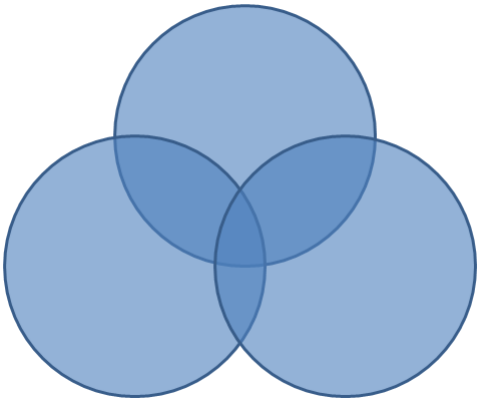

Figure 2: Area of union is bounded by sum of areas of the circles.

Our fifth essential analysis tool is the *union bound*, which is not a tail inequality but is often used in conjunction with tail inequalities. The union bound just says that for events $E_1, \ldots, E_k$,

$$\mathbf{Pr}[\text{at least once of } E_i \text{ occurs}] \leq \sum_{i=1}^{k} \mathbf{Pr}[E_i]\,.$$

Importantly, the events are completely arbitrary, and do not need to be independent. The proof is a one-liner. In terms of Figure 2, the union bound just says that the area (i.e., probability mass) in the union is bounded above by the sum of the areas of the circles. The bound is tight if the events are disjoint; otherwise the right-hand side is larger, due to double-counting. (It's like inclusion-exclusion, but without any of the correction terms.) In

applications, the events $E_1, \ldots, E_k$ are often "bad events" that we're hoping don't happen; the union bound says that as long as each event occurs with low probability and there aren't too many events, then with high probability none of them occur.

Returning to our running hashing example, let E_i denote the event that bucket i receives a load larger than $3 \ln n / \ln \ln n$. Using (4) and the union bound, we conclude that with probability at least $1 - \frac{1}{n}$, *none* of the buckets receive a load larger than $3 \ln n / \ln \ln n$. That is, the maximum load is $O(\log n / \log \log n)$ with high probability.[5]

3.6 Chernoff Bounds: The Large Expectation Regime

We previously noted that the Chernoff bounds yield very good probability bounds once the deviation $(1+\delta)$ or the expectation ($\mathbf{E}[X]$) becomes large. In our hashing application above, we were in the former regime. To illustrate the latter regime, suppose that we hash a data set $S \subseteq U$ with $|S| = n \ln n$ (instead of $\ln n$). Now, the expected load of every bucket is $\ln n$. Applying Theorem 3.3 with $1 + \delta = 4$, we get that, for each bucket i,

$$\mathbf{Pr}[\text{load on } i \text{ is } > 4 \ln n] \leq \left(\frac{e}{4}\right)^{4 \ln n} \leq \frac{1}{n^2}.$$

Using the union bound as before, we conclude that with high probability, no bucket receives a load more than a small constant factor times its expectation.

Summarizing, when loads are light there can be non-trivial deviations from expected loads (though still only logarithmic). Once loads are even modestly larger, however, the buckets are quite evenly balanced with high probability. This is a useful lesson to remember, for example in load-balancing applications (in data centers, etc.).

4 Randomized Rounding

We now return to the design and analysis of approximation algorithms, and give a classic application of the Chernoff bounds to the problem of low-congestion routing.

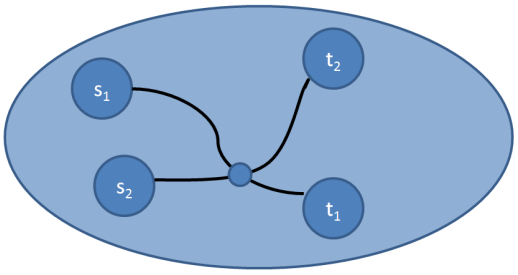

Figure 3: Example of edge-disjoint path problem. Note that vertices can be shared, as shown in this example.

[5]There is also a matching lower bound (up to constant factors).

If the *edge-disjoint paths* problems, the input is a graph $G = (V, E)$ (directed or undirected) and source-sink pairs $(s_1, t_1), \ldots, (s_k, t_k)$. The goal is to determine whether or not there is an s_i-t_i path P_i for each i such that no edge appears in more than one of the P_i's. See Figure 3. The problem is NP-hard (for directed graphs, even when $k = 2$).

Recall from last lecture the linear programming rounding approach to approximation algorithms:

1. Solve an LP relaxation of the problem. (For an NP-hard problem, we expect the optimal solution to be fractional, and hence not immediately meaningful.)

2. "Round" the resulting fractional solution to a feasible (integral) solution, hopefully without degrading the objective function value by too much.

Last lecture applied LP rounding to the vertex cover problem. For the edge-disjoint paths problem, we'll use *randomized* LP rounding. The idea is to interpret the fractional values in an LP solution as specifying a probability distribution, and then to round variables to integers randomly according to this distribution.

The first step of the algorithm is to solve the natural linear programming relaxation of the edge-disjoint paths problem. This is just a multicommodity flow problem (as in Exercise Set #5 and Problem Set #3). In this relaxation the question is whether or not it is possible to send simultaneously one unit of (fractional) flow from each source s_i to the corresponding sink t_i, where every edge has a capacity of 1. 0-1 solutions to this multicommodity flow problem correspond to edge-disjoint paths. As we've seen, this LP relaxation can be solved in polynomial time. If this LP relaxation is infeasible, then we can conclude that the original edge-disjoint paths problem is infeasible as well.

Assume now that the LP relaxation is feasible. The second step rounds each s_i-t_i pair independently. Consider a path decomposition (Problem Set #1) of the flow being pushed from s_i to t_i. This gives a collection of paths, together with some amount of flow on each path. Since exactly one unit of flow is sent, we can interpret this path decomposition as a probability distribution over s_i-t_i paths. The algorithm then just selects an s_i-t_i path randomly according to this probability distribution.

The rounding step yields paths $P_1, \ldots, P_k$. In general, they will not be disjoint (this would solve an NP-hard problem), and the goal is to prove that they are approximately disjoint in some sense. The following result is the original and still canonical application of randomized rounding.

Theorem 4.1 *Assume that the LP relaxation is feasible. Then with high probability, the randomized rounding algorithm above outputs a collection of paths such that no edge is used by more than*

$$\frac{3 \ln m}{\ln \ln m}$$

of the paths, where m is the number of edges.

The outline of the proof is:

1. Fix an edge e. The expected number of paths that include e is at most 1. (By linearity of expectation, it is precisely the amount of flow sent on e by the multicommodity flow relaxation, which is at most 1 since all edges were given unit capacity.)

2. Like in the hashing analysis in Section 3.6,

$$\mathbf{Pr}\left[\#\text{ paths on } e > \frac{3\ln m}{\ln\ln m}\right] \leq \frac{1}{m^2},$$

   where m is the number of edges. (Edges are playing the role of buckets, and s_i-t_i pairs as items.)

3. Taking a union bound over the m edges, we conclude that with all but $\frac{1}{m}$ probability, every edge winds up with at most $3\ln m/\ln\ln m$ paths using it.

Zillions of analyses in algorithms (and theoretical computer science more broadly) use this one-two punch of the Chernoff bound and the union bound.

Interestingly, for directed graphs, the approximation guarantee in Theorem 4.1 is optimal, up to a constant factor (assuming $P \neq NP$). For undirected graphs, there is an intriguing gap between the $O(\log n/\log\log n)$ upper bound of Theorem 4.1 and the best-know lower bound of $\Omega(\log\log n)$ (assuming $P \neq NP$).

5 Epilogue

To recap the top 5 essential tools for the analysis of randomized algorithms:

1. *Linearity of expectation.* If all you care about is the expectation of a random variable, this is often good enough.

2. *Markov's inequality.* This inequality usually suffices if you're satisfied with a constant-probability bound.

3. *Chebyshev's inequality.* This inequality is the appropriate one when you have a good handle on the variance of your random variable.

4. *Chernoff bounds.* This inequality gives sharp concentration bounds for random variables that are sums of independent and bounded random variables (most commonly, sums of independent indicator random variables).

5. *Union bound.* This inequality allows you to avoid lots of bad low-probability events.

All five of these tools are insanely useful. And four out of the five have one-line proofs!