Colorado CSCI 5454: Algorithms **Fall 2020**

Randomized Algorithms

Instructor: Bo Waggoner *Lecture 8*

This lecture will first review basic probability, then introduce algorithms that use randomness to solve (or approximately solve) problems.

Objectives:

- Be comfortable with basics of probability; be able to analyze randomized algorithms.
- Know how approximation ratios and factors can be applied to randomized algorithms (i.e. expected value of the solution).
- Learn some examples of randomized algorithms in cases where deterministic solutions are difficult.

1 Probability review

In this review, we will focus on finite, discrete sample spaces such as coin flips and die rolls. Later in the class we will see real-valued random variables such as Gaussians, but we won't need to fully review measure theory in order to analyze algorithms in this class.

Probability space. We have a **sample space** is a finite (for now) set Ω of possible *outcomes* (things that might happen). We have a **distribution** p assigning a number $p(\omega) \in [0,1]$ for each $\omega \in \Omega$ such that $\sum_{\omega \in \Omega} p(\omega) = 1$.

Example: we roll two fair, independent die, so the possible outcomes are $\Omega = \{(1,1),(1,2),\cdots,(6,6)\}$. The distribution is $p(\omega) = \frac{1}{36}$ for all ω.

An **event** is a subset of outcomes. For example, the event that the sum of the die is 11 or higher is $A = \{(5,6),(6,5),(6,6)\}$. Because we have a finite discrete sample space, $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$. In the example, $\Pr[A] = \frac{3}{36}$.

Joint probability. The event that A and B both occur is $A \cap B$. This is an intersection of sets, yielding some other subset of Ω. We can also write "A and B".

For example: the event that first roll is a 6 is $B = \{(6,1),(6,2),\dots,(6,6)\}$. The event that sum is 11 or higher *and* the first roll is a 6 is: $A \cap B = \{(6,5),(6,6)\}$. The probability of both A and B is $\Pr[A \cap B] = \Pr[A \text{ and } B] = \frac{2}{36}$.

The event that either A occurs, or B occurs, or both, is $A \cup B$. We can also write "A or B". The event that sum is 11 or higher *or* first roll is a 6 is: $A \cup B = \{(6,1),(6,2),(6,3),(6,4),(6,5),(6,6),(5,6)\}$. The probability of this event is $\Pr[A \cup B] = \Pr[A \text{ or } B] = \frac{7}{36}$.

Conditional probability. The probability of event B given that A occurs, for $\Pr[A] > 0$, is written $\Pr[B \mid A] := \frac{\Pr[A \cap B]}{\Pr[A]}$.

Example: the probability the first roll is six given the sum is at least 11 is

$$\begin{aligned}\Pr[B \mid A] &= \frac{\Pr[A \cap B]}{\Pr[A]} \\ &= \frac{2/36}{3/36} \\ &= \frac{2}{3}.\end{aligned}$$

Independence. Events A and B are **independent** if $\Pr[A \cap B] = \Pr[A]\Pr[B]$. In other words, the probability of both A and B occurring is the product of their probabilities. For example, if B is the event that the first die is a 6 and C is the event that the second die is a 6, then B and C are independent: $\Pr[B] = \Pr[\{(6,1),\dots,(6,6)\}] = \frac{1}{6}$, and a similar calculation shows $\Pr[C] = \frac{1}{6}$; and $\Pr[B \cap C] = \Pr[\{(6,6)\}] = \frac{1}{36} = \left(\frac{1}{6}\right)\left(\frac{1}{6}\right)$.

Exercise 1. Check that if two events A and B are independent and $\Pr[B] > 0$, then $\Pr[A \mid B] = \Pr[A]$. (Recall the definition of $\Pr[A \mid B]$.)

Exercise 2. Let D be the event that the sum of the dice is odd and let E be the event that the first die is at most 4. Show that D and E are independent.

1.1 Random variables

Formally, a **random variable (RV)** is a function $X : \Omega \to \mathbb{R}$. (The range doesn't have to be the real numbers, but usually it is.) Continuing the two-dice example, write $\omega = (a, b)$ for the outcomes of the dice. An example RV is $X(a, b) = a + b$, the sum of the two dice.

Recall that events are subsets of the sample space. We can now write events as e.g. $X \geq 11$, which technically means the event $\{\omega : X(\omega) \geq 11\}$. This is the same as event A in the previous example: the sum of the dice is at least 11. So we can write $\Pr[X \geq 11]$.

Example: Let $Y(a, b) = a$, i.e. Y is the value of the first die. Then $\Pr[Y = 6 \mid X \geq 11] = \frac{2}{3}$.

If we take a function of random variables, e.g. $5X^2 + Y + 2$, this is another random variable, and we can talk about probabilities of events like $\Pr[5X^2 + Y + 2 \geq 10]$.

Random variables X, Y are **independent** if for all values x, y, $\Pr[X = x \text{ and } Y = y] = \Pr[X = x]\Pr[Y = y]$. For example, if Y is the value of the first die and Z the second die, then Y and Z are independent.

The **expectation** of a random variable is $\mathbb{E}[X] := \sum_x x \Pr[X = x]$. The sum ranges over all possible values x of X. We can also write it as $\mathbb{E}[X] = \sum_{\omega \in \Omega} p(\omega) X(\omega)$.

Examples above:

$$\begin{aligned}\mathbb{E}[Y] &= \sum_{y=1}^{6} \Pr[Y = y] \cdot y \\ &= \frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) \\ &= 3.5.\end{aligned}$$

Similarly, for $X =$ the sum of the dice, $\mathbb{E}[X] = 7$. Again, we can consider random variables that are functions of others, e.g. $\mathbb{E}[5X^2 + Y + 2]$.

Fact 1 (Linearity of expectation)**.** *For any random variables X, Y, we have $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$.*

Proof.

$$\begin{aligned}\mathbb{E}[X+Y] &= \sum_{\omega\in\Omega} p(\omega)\left(X(\omega)+Y(\omega)\right) \\ &= \left(\sum_{\omega\in\Omega} p(\omega)X(\omega)\right) + \left(\sum_{\omega\in\Omega} p(\omega)Y(\omega)\right) \\ &= \mathbb{E}[X] + \mathbb{E}[Y].\end{aligned}$$

□

This is great because it's true no matter what, even if X and Y are correlated.

Conditional expectations. Consider an event, such as $Y = y$, with nonzero probability. We define the conditional expectation of X, conditioned on this event, as:

$$E[X \mid Y = y] := \sum_x \Pr[X = x \mid Y = y] x.$$

We may use subscripts to denote an expectation taken only over a particular random variable:

$$\mathop{\mathbb{E}}_{Y}\mathop{\mathbb{E}}_{X}[X \mid Y] := \sum_y \Pr[Y = y]\left(\sum_x \Pr[X = x \mid Y = y] x\right).$$

Another important probability fact:

Fact 2 (Law of total expectation)**.** *Let X, Y be random variables. Then $\mathbb{E}[X] = \mathbb{E}_Y\left(\mathbb{E}_X[X \mid Y]\right)$.*

Proof.

$$\begin{aligned}\mathop{\mathbb{E}}_{Y}\left(\mathop{\mathbb{E}}_{X}[X \mid Y]\right) &= \sum_{x,y} \Pr[X = x \text{ and } Y = y] x \\ &= \sum_x x \Pr[X = x]\left(\sum_y \Pr[Y = y \mid X = x]\right) \\ &= \sum_x x \Pr[X = x]\end{aligned}$$

□

Given an event A, the **indicator random variable $\mathbf{1}[A]$** is $\mathbf{1}[A](\omega) = 1$ if $\omega \in A$, otherwise 0.

Exercise 3. Show that if X is a random variable and $\beta \in \mathbb{R}$, then $\mathbb{E}[\beta X] = \beta\,\mathbb{E}[X]$.

Exercise 4. Show that for any event A, $\mathbb{E}[\mathbf{1}[A]] = \Pr[A]$.

2 Randomized Algorithms

We will formalize randomized algorithms via the word RAM model, where we add an "oracle", i.e. a function or API the algorithm can query. We suppose that the algorithm can, in constant time, ask for and receive an independent random number. For our purposes, this can be a bit (i.e. zero or one); a uniformly random integer in a finite range; or even a uniformly random real number in $[0,1]$. (In some contexts, one may rightly worry about whether this is realistic.)

For randomized algorithms, we need to modify our notions of **efficiency** and **correctness**. Generally, for efficiency, we will still measure the worst-case running time, regardless of the randomness. However, in some cases the average-case or expected runtime is also interesting. The same goes for space usage.

There are several notions of correctness. We will not focus much on the first two.

- The algorithm produces the correct answer with at least a certain probability. This is often called a **Monte Carlo** algorithm; we will see an example with global min cut.
- The algorithm is always correct, although its running time may be random. This is often called a **Las Vegas** algorithm; an example is randomized quicksort.
- The algorithm **approximates** the answer to an optimization problem. We will discuss this next.
- Algorithms to produce a **randomly distributed output** for some use, like a sample from a distribution. We will see this later in the course.

2.1 Randomized optimization algorithms

For an optimization problem of the form $\max_{S \in \mathcal{F}} f(S)$, a randomized algorithm's performance ALG $= f(S)$ is a random variable. The most natural way to measure it is by its expectation. (Of course, one can also ask for other guarantees, such as a good objective value with high probability.)

For a maximization problem $\max_{\vec{x} \in \mathcal{F}} f(\vec{x})$, the algorithm has an **approximation ratio** α if $\mathbb{E}[\text{ALG}] \geq \alpha \cdot \text{OPT}$. Similarly, for a minimization problem $\min_{\vec{x} \in \mathcal{F}} f(\vec{x})$, it achieves a C**-approximation** if $\mathbb{E}[\text{ALG}] \leq C \cdot \text{OPT}$.

3 Max-3SAT

A 3-CNF is a boolean formula in conjunctive normal form (CNF) with 3 literals per clause. Let us unpack what this means.

- We have n variables $x_1, \ldots, x_n$ which are boolean (can be set to true or false).
- A *literal* is either a variable x_i or its negation $\bar{x_i}$, meaning NOT x_i.
- A *clause* in this case is an OR of three literals, e.g. x_1 OR $\bar{x_2}$ OR x_3, where a variable cannot appear twice. Note the clause is true if any of the three literals evaluates to true, otherwise it is false.
- A 3-CNF formula is an AND of a sequence of m clauses, e.g.

$$(x_i \text{ OR } x_j \text{ OR } \bar{x_k}) \text{ AND } \cdots \text{ AND } (x_a \text{ OR } \bar{x_b} \text{ OR } x_c)$$

  This evaluates to TRUE if *all* of the m clauses evaluate to TRUE.

Determining if a given 3-CNF is satisfiable, i.e. if there exists a setting of $x_1, \ldots, x_n$ such that every clause is TRUE, is NP-complete.

The **Max-3SAT problem** is:

- Input a 3-CNF formula on n variables, m clauses.
- Output: a setting of $x_1, \ldots, x_n$ to maximize the number of TRUE clauses.

Theorem 1. *Given any Max-3SAT instance, there always exists a choice of variables such that $\geq \frac{7}{8}$ of the clauses are TRUE.*

To prove it, we consider the following **Uniform-Random algorithm:** For each x_i, independently set it to be TRUE with half probability, FALSE otherwise.

Proposition 1. *In expectation, under Uniform-Random, at least $\frac{7}{8}m$ clauses are TRUE.*

Proof. Consider any clause $j \in \{1, \dots, m\}$. Let the random variable $Z_j = 1$ if it is satisfied, 0 otherwise.

Lemma 1. *For any j,* $\Pr[Z_j = 1] = \frac{7}{8}$.

Exercise 5. Prove Lemma 1.

Let Y be the number of satisfied clauses, i.e. $Y = \sum_{j=1}^{m} Z_j$. Then the expected number of satisfied clauses is

$$\mathbb{E}[Y] = \mathbb{E}\left[\sum_{j=1}^{m} Z_j\right] = \sum_{j=1}^{m} \mathbb{E}[Z_j] = m \cdot \frac{7}{8}.$$

We used linearity of expectation. Then we used that, as a indicator variable, $\mathbb{E}[Z_j] = \Pr[Z_j = 1] = \frac{7}{8}$. □

Now, we can use the algorithm to prove the existence result.

Proof of Theorem 1. Let Y be the number of satisfied clauses when choosing an assignment uniformly at random. By Proposition 1, $\mathbb{E}[Y] \geq \frac{7}{8}m$. Therefore, $\Pr[Y \geq \frac{7}{8}m] > 0$. (Otherwise, the expectation would be strictly less than zero.) In other words, there exists some assignment to $x_1, \dots, x_n$ such that at least $\frac{7}{8}$ of the clauses are satisfied. □

This is an example of the *probabilistic method*: prove something exists by randomly constructing it, and showing this random construction sometimes succeeds or succeeds on average.

We also get this corollary:

Corollary 1. *Uniform-Random has an approximation ratio of $\frac{7}{8}$.*

Proof. We already showed that $\mathbb{E}[\text{ALG}] \geq \frac{7}{8}m$. But $\text{OPT} \leq m$, since there are only m clauses total. So $\mathbb{E}[\text{ALG}] \geq \frac{7}{8}\text{OPT}$. □

Of course, Uniform-Random is a very naive algorithm, and you might hope for a better one. But amazingly, the following is known:

Theorem 2 (Håsted). *If $P \neq NP$, then no polynomial-time algorithm guarantees better than a $\frac{7}{8}$ approximation ratio for Max-3SAT.*

Exercise 6. In one variant of Max-3SAT, clauses do not have to contain exactly 3 literals, but can contain up to 3 literals. Is Uniform-Random still a $\frac{7}{8}$ approximation to this problem? Why or why not? *Hint: start by considering examples with just one clause.*

4 Derandomizing the Max-3SAT Algorithm

It is often of interest to take a randomized algorithm and convert it into a deterministic one, a process called *derandomization.* As we will see next, the idea is often to consider each randomized choice and make it in a way that preserves the performance guarantee of the algorithm.

Proposition 2. *There is a deterministic polynomial-time $\frac{7}{8}$ approximation algorithm for Max-3SAT.*

Proof. The algorithm and proof are as follows. First consider Uniform-Random, but imagine $x_1 =$ TRUE. Compute $\mathbb{E}[m|x_1 = \text{TRUE}]$, the expected performance conditioned on setting x_1 to TRUE. We can compute this easily: for clauses without x_1, the probability of being satisfied is still $\frac{7}{8}$; for clauses containing the literal x_1, the probability is 1; and for clauses containing $\bar{x_1}$, it is $\frac{3}{4}$ (as there are two remaining literals). We can sum these over the clauses in linear time.

Now if $\mathbb{E}[m \mid x_1 = \text{TRUE}] \geq \frac{7}{8}$, we set $x_1 =$ TRUE. Otherwise, we set $x_1 =$ FALSE, and we claim that in this case, $\mathbb{E}[m \mid x_1 = \text{FALSE}] \geq \frac{7}{8}$. The reason is that at least one of these two quantities must exceed their average of $\frac{7}{8}$:

$$\begin{aligned}\frac{7}{8} &= \mathbb{E}[m] \\ &= \Pr[x_1 = \text{TRUE}]\, \mathbb{E}[m \mid x_1 = \text{TRUE}] + \Pr[x_1 = \text{FALSE}]\, \mathbb{E}[m \mid x_1 = \text{FALSE}] \\ &= \frac{1}{2}\, \mathbb{E}[m \mid x_1 = \text{TRUE}] + \frac{1}{2}\, \mathbb{E}[m \mid x_1 = \text{FALSE}].\end{aligned}$$

So they cannot both be strictly less than $\frac{7}{8}$.

Having finalized x_1, we are left with a set of clauses each with 2 or 3 literals, where the expected performance of Uniform-Random (choosing the remaining variables) is at least $\frac{7}{8}$. So we can again compute a choice of x_2 such that the conditional expectation remains above $\frac{7}{8}$.

Repeating the argument, we eventually choose all variables $x_1, \ldots, x_n$, and are left with at least $\frac{7}{8}$ of the clauses are satisfied. Furthermore, this was a deterministic algorithm, as claimed, and it ran in approximately quadratic time: in each of n iterations, it needed $O(n + m)$ computation to determine the conditional expectation. □

5 Min Weighted Vertex Cover

Given an undirected graph $G = (V, E)$, a **vertex cover** is a set of vertices $S \subseteq V$ that "cover" all the edges, i.e. for all $\{u, v\} \in E$, at least one of the endpoints $\{u, v\}$ is in S.

In the **weighted vertex cover problem**, the input is an undirected graph $G = (V, E)$ and a list of positive vertex weights, w_v for each $v \in V$. The goal is to output a vertex cover S with smallest **total weight** $w(S) := \sum_{v \in S} w_v$.

Even if all the weights are 1, solving this problem exactly is NP-hard. However, we will see an algorithm finding a vertex cover with expected total weight at most twice OPT.

Algorithm: Initialize $S = \emptyset$. For each edge $e = \{u, v\}$:

- If either u or v is already in S, continue.
- Independently with probability $\frac{w_v}{w_u + w_v}$, add u to S. Otherwise, add v. (Note the probability of adding v is $\frac{w_u}{w_u + w_v}$.)

Note the idea is to bias toward adding the *smaller*-weight vertex: If w_v is much larger than w_u, then we are much more likely to add u.

We will skip the running time and space analysis. The key idea for approximation will be that most of the "weight" in S comes from OPT vertices, so S cannot be much larger than OPT.

Theorem 3. *This randomized algorithm gives a 2-approximation.*

Proof. First, note that the algorithm does always give a vertex cover, because we iterate through all edges and, if not yet covered, always add one of its endpoints to S.

Let $S_0, \ldots, S_m$ be the sets of the algorithm at each round, with $S_0 = \emptyset$. Define $A_t = S_t \cap \text{OPT}$. In other words, it is all of the vertices added so far that are also in the optimal smallest-weight vertex cover. Meanwhile, define $B_t = S_t \setminus A_t$. This is all of the non-OPT vertices that are added.[1]

[1]Notice the algorithm doesn't know these things because it doesn't know OPT. We are only using them in the analysis.

We will prove in Lemma 2 that $\mathbb{E}[w(B_m)] \leq \mathbb{E}[w(A_m)]$. In other words, S contains more weight from OPT than weight from non-OPT vertices. Using this,

$$\begin{aligned}\mathbb{E}[w(S)] &= \mathbb{E}[w(A_m)] + \mathbb{E}[w(B_m)] \\ &\leq 2\,\mathbb{E}[w(A_m)] \\ &\leq 2w(\mathrm{OPT}).\end{aligned}$$

The last line follows because $A_m \subseteq \mathrm{OPT}$, so $w(A_m) \leq w(\mathrm{OPT})$. This proves the claim. □

Now we need to prove Lemma 2.

Lemma 2. $\mathbb{E}[w(B_m)] \leq \mathbb{E}[w(A_m)]$.

Proof. Let $X_t = w(A_t) - w(A_{t-1})$. Let $Y_t = w(B_t) - w(B_{t-1})$. These are random variables, the increases in the weights of the sets at each round. Now, we will need one more fact, Lemma 3: at each round t, if we fix the outcomes of the previous rounds $S_{t-1} = s$, then $\mathbb{E}[X_t \mid S_{t-1} = s] \geq \mathbb{E}[Y_t \mid S_{t-1} = s]$.

Given this fact, we know that $\mathbb{E}[Y_t] \leq \mathbb{E}[X_t]$ by the law of total expectation, because $\mathbb{E}[X_t] = \mathbb{E}_{S_{t-1}} \mathbb{E}[X_t \mid S_{t-1}]$ and similarly for Y_t. So

$$\begin{aligned}\mathbb{E}[w(B_m)] &= \mathbb{E}\left[\sum_{t=1}^{m} Y_t\right] \\ &= \sum_{t=1}^{m} \mathbb{E}[Y_t] \\ &\leq \sum_{t=1}^{m} \mathbb{E}[X_t] \\ &= \mathbb{E}[w(A_m)].\end{aligned}$$

□

Lemma 3. *For each round t and fixed $S_{t-1} = s$, $\mathbb{E}\left[X_t \mid S_{t-1} = s\right] \geq \mathbb{E}\left[Y_t \mid S_{t-1} = s\right]$.*

Proof. At round t, if the edge is already covered by some vertex in S_{t-1}, the sets remain the same, i.e. $\mathbb{E}[X_t \mid S_{t-1} = s] = \mathbb{E}[Y_t \mid S_{t-1} = s] = 0$. If the edge is not yet covered, then OPT must contain either u, or v, or both. If it contains u only, then

$$\begin{aligned}\mathbb{E}[X_t \mid S_{t-1} = s] &= \Pr[\text{add } u] w_u \\ &= \frac{w_u w_v}{w_u + w_v} \\ &= \Pr[\text{add } v] w_v \\ &= \mathbb{E}[Y_t \mid S_{t-1} = s].\end{aligned}$$

If OPT contains v only, then the expected increases are exactly the same. (Check.) And if OPT contains both u and v, then $\mathbb{E}[Y_t \mid S_{t-1} = s] = 0$ while $\mathbb{E}[X_t \mid S_{t-1} = s] > 0$. This proves that for any outcome of S_{t-1}, the inequality holds. □

Exercise 7. What is the size of the largest possible vertex cover of a graph $G = (V, E)$? *(Recall the definition of vertex cover.)*

Exercise 8. Suppose all the vertex weights are 1. Show that even in this case, Theorem 3 is tight by analyzing an instance where $\mathbb{E}[\mathrm{ALG}] = 2\,\mathbb{E}[\mathrm{OPT}]$, or a family of instances where $\mathbb{E}[\mathrm{ALG}] \to 2E[\mathrm{OPT}]$. *Hint: consider a star graph.*

6 Min-cut contraction algorithm

Exercise 9. Recall: what is a definition of a **cut** S, T in a graph $G = (V, E)$?

In the **global min cut problem**, the input is an undirected, unweighted graph $G = (V, E)$. In this case, the **value** of a cut (S, T) is the number of edges crossing it, i.e. $|\{\{u, v\} \in E : u \in S, v \in T\}|$.

Call a cut **trivial** if $S = \emptyset$ or $T = \emptyset$. Naturally, the value of a trivial cut is zero (agreed?). So the goal is to output is a nontrivial cut with minimum value.

Exercise 10. Suppose the graph has a vertex of degree k. Argue that the global min cut is at most k.

Exercise 11. Suppose the average degree in the graph is k. Again argue that the global min cut is at most k. *Hint: reduce to the previous exercise.*

Note that one polynomial-time approach is to consider all possible pairs of vertices s, t. For each, we convert the graph to a directed graph with capacity 1 in both directions on each edge; then find the min s-to-t cut. The smallest of these would be the global min cut.

Here, we will consider a randomized algorithm that finds the *exact* correct solution with some probability. We will assume the graph is connected. Otherwise, there is a cut with value 0 that can be easily found in linear time.

Exercise 12. In undirected graph, describe a linear-time algorithm to check if it is connected and, if not, to return a cut with value 0. *Hint: use a graph search algorithm.*

Algorithm. For our randomized algorithm, it will be useful to define a *multigraph*: graph that may have multiple edges between any pair of vertices. In this case, we simply let E be a *multiset* instead of a set: it may contain a given edge $\{u, v\}$ multiple times, rather than only zero or one times.

Now, define **contracting** an undirected multigraph $G = (V, E)$ **along an edge** $e = \{u, v\}$. We create a copy of the graph, G', except that we do not include u, v, or any of their incident edges. Then we add a new vertex a_{uv} that represents the "joining" of these two. For every edge $\{u', u\}$ or $\{u', v\}$ in the original graph, we add an edge $\{u', a_{uv}\}$ in the new graph.

Note that edges are still undirected and the new graph may also be a multigraph. We say that u and v have been *contracted into* a_{uv} and so on.

Exercise 13. Take a complete graph on four vertices (i.e. 6 edges) and contract it three times, each time re-using the vertex created at the previous step. How many edges are in the multigraph after the first, second, and third contraction?

Now we can define Karger's **Contraction Algorithm**:

- Pick an edge uniformly at random.
- Contract the graph along that edge.
- Repeat until only two vertices u, v remain.
- Return the cut corresponding to these two vertices (i.e. U_1 = all vertices that were contracted into u, U_2 = all that were contracted into v).

Theorem 4. *The probability of correctness of the Contraction Algorithm is at least* $\frac{2}{n(n-1)}$.

Note: this is not very good, but one can show that repeating the algorithm polynomially many times, and picking the smallest cut found, gives a good success probability.

Proof. Let k be the size of the min cut (pick a particular min cut if there are more than one). Let E_t be the event that that, in round t, the edge selected does *not* belong to the min cut. There are $n-2$ rounds, because in each round we decrease the number of vertices by one until we have two left. The algorithm outputs the min cut if it never contracts any edge in the min cut (agreed?), so the goal is to prove that $\Pr[E_1 \cap \cdots \cap E_{n-2}] \geq \frac{2}{n(n-1)}$.

Note each vertex has degree at least k, otherwise putting it in U_1 alone would be a smaller cut. So the total number of edge endpoints is at least nk where n is the number of vertices. So the number of edges is at least $\frac{nk}{2}$.

We pick an edge uniformly at random, and there are at least $\frac{nk}{2}$ edges, so $\Pr[E_1] \geq 1 - \frac{k}{|E|} \geq 1 - \frac{2}{n}$. If we succeeded in the first round, we have an $n-1$ vertex graph with min cut k, so by the same reasoning, $\Pr[E_2 \mid E_1] \geq 1 - \frac{2}{n-1}$. Repeat until n drops to 2: we have

$$\begin{aligned}
\Pr[E_1 \cap \cdots \cap E_{n-2}] &= \Pr[E_1] \Pr[E_2 \mid E_1] \cdots \Pr[E_{n-2} \mid E_1 \cap \cdots \cap E_{n-3}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right) \cdots \left(\frac{1}{3}\right) \\
&= \frac{(n-2)!(2)}{n!} \\
&= \frac{2}{n(n-1)}.
\end{aligned}$$

□

Exercise 14. Suppose we picked a cut completely at random. What would be the probability of finding the min cut (suppose there is only one min cut) and how does this compare to the contraction algorithm?