Randomized Algorithms

So far, we have been mostly concerned with the worst-case analysis of algorithm run times. In many scenarios, it is more informative to worry about average case run-time. We can do that in terms of analyzing the sample space of the input, or by randomizing the algorithm in some way. We start with the latter. There are two main types of random algorithms (and combinations of the two): Las Vegas, which are always correct and fast with high probability, and Monte Carlo, which are always fast and correct with high probability.
First, a few definitions and useful facts about random variables and expectation. Let X be a random variable, and $\mathbb{P}(X = x)$ the probability that X takes value x. We will just deal with discrete random variables in this class: ones that take values from a countable set, usually natural numbers. Then the *expected value* of X is defined as

$$\mathbb{E}(X) = \sum_{x \in S} x\mathbb{P}(X = x).$$

We let I_A be an *indicator random variable* for the event A. This means that

$$I_A = \begin{cases} 1, & \text{if } A \text{ occurs} \\ 0, & \text{otherwise.} \end{cases}$$

Here, it is helpful to notice that

$$\mathbb{E}(I_A) = \mathbb{P}(A).$$

Finally, we will use linearity of expectation:

Lemma 1. *We have* linearity of expectation*:*

$$\mathbb{E}\left(\sum_i X_i\right) = \sum_i \mathbb{E}(X_i),$$

whether the variables X_i are dependent or not.

Quicksort

This is not a graph problem, but it is a nice example of the power of a randomized algorithm (Las Vegas). We take as input a list of elements (with some total ordering on them), and wish to return a sorted list. The quicksort algorithm, *qs* for short, is as follows: if the list has length 0 or 1 return the list itself. Else, select an element x (called a pivot), and divide the remaining elements into two lists L_1 and L_2, such that all elements in L_1 are less than x and all elements in L_2 greater than x. Then, perform quicksort on L_1 and L_2 and return $qs(L_1), x, qs(L_2)$ concatenated.

If we look at the runtime, we have to consider the worst case. If the list is already sorted, or if for some reason we select the pivots exactly in ascending order, then every element is compared to every element, and thus the best we can say about the run time is that it is $O(n^2)$.

Now, suppose that we select the pivot uniformly at random each time (meaning that every element is equally likely to be selected). Instead of looking at the worst-case run time, we

look at the average, or expected, run time. First, note that the order of the elements in the input list is now irrelevant for the run time. Assume the elements are $1, \ldots, n$. We let X_{ij}, for $i < j$ be the indicator random variable for the event that element i and element j are compared during the execution of quicksort. Note that whether i and j are ever compared depends on the first time that a pivot p is chosen from the interval $i, i+1, \ldots, j$. If this pivot p is such that $i < p < j$, then i and j are separated and never compared. If $p \in \{i, j\}$, they are compared. Therefore,

$$\mathbb{P}(i \text{ and } j \text{ are compared}) = \frac{2}{j-i+1}.$$

Now note that the number of comparisons Y is simply the sum of all the indicator random variables X_{ij}, and

$$\mathbb{E}(Y) = \mathbb{E}\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbb{E}(X_{ij}) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbb{P}(i \text{ and } j \text{ are compared}),$$

by linearity of expectation and the properties of indicator random variables. So, we have

$$\mathbb{E}(Y) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1}.$$

For a given i, we have

$$\sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-i+1}\right) = O(\log n),$$

as this is the Harmonic Series. This gives us

$$\mathbb{E}(Y) = O(n \log n).$$

Exercise 1. *Suppose that we make the following improvement. In each round, select 3 candidates for a pivot uniformly at random, i.e. select a random triple from all $\binom{n}{3}$ possible triples, and then let the final pivot be the middle of the three. Show that this gives a smaller $\mathbb{P}(i$ and j are compared$)$ when $i < j-1$ (note when $i = j-1$ we have $X_{ij} = 1$ always). You don't have to complete the full analysis.*

Karger's algorithm for minimum cut

We have seen the usefulness of minimum cuts in various applications (e.g. finding efficient tree decompositions), and we showed that minimum cuts can be found in polynomial time. However, we were not too precise about how quickly they can be found exactly. Karger's algorithm is an extremely elegant randomized approach (Monte Carlo). The idea of the algorithm is as follows: if an edge is chosen at random, then its endpoints are more likely to lie on the same side of a minimum cut than acorss from it. Indeed, the point of a min cut is that the vertices on each side are "clustered" (have many edges between them), with sparse connection going across the cut. The algorithm repeatedly contracts edges until there are only two vertices left which represent two clusters across some cut. Then, we argue that this resulting cut is likely small.

For the sake of simplicity, we will look at the probability that the algorithm finds a particular minimum cut, say C. (If there are multiple minimum cuts, that would only improve its efficiency.) For this particular algorithm, we use edge contractions which do not keep self-loops, but do keep multiple edges. Take a graph G, select an edge e uniformly at random and contract it to obtain the graph G/e. If e was not in C, then the C is a mincut of G/e. If we are lucky, this process continues until our graph consists of two vertices with C across them. We note that if $\delta(G)$ is the minimum degree of G then $|C| \leq \delta$, since we can always seperate a vertex of minimum degree from the rest of the graph. This is also true for the average degree: $|C| \leq 2|E|/n$, where $E = E(G)$. Therefore, if we select an edge e uniformly at random from E, we have

$$\mathbb{P}(e \in C) \leq \frac{2|E|/n}{|E|} = \frac{2}{n}.$$

Note that at every step of the algorithm, the number of edges decreases by varying numbers, but the number of vertices always decreases by exactly one. The probability of success of the algorithm (finding C) is exactly the probability that in each round, we select an edge that is not in C. Therefore, we have

$$\begin{aligned}\mathbb{P}(C \text{ is found}) &\geq (1-\frac{2}{n})(1-\frac{2}{n-1})(1-\frac{2}{n-2})\dots(1-\frac{2}{3}) \\ &= \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdot\frac{n-5}{n-3}\cdot\dots\cdot\frac{2}{4}\cdot\frac{1}{3} \\ &= \frac{2}{n(n-1)}.\end{aligned}$$

This means that in expectation, the minimum cut is found after $\Omega(n^2)$ rounds. One can then decide on how many runs to use depending on the desired level of confidence.

Here is another way to think of Karger's algorithm: assign random weights to the edges and find a minimum spanning tree T. Then the largest edge in T corresponds to the cut output by the algorithm.

Exercise 2. *Give examples of a few classes of graphs where $\mathbb{P}(C$ is found$) = 1$ always.*

Exercise 3. *Find a nontrivial upper bound on $\mathbb{P}(C$ is found$)$ in the case of complete graphs.*