Lecture 14: Randomized Complexity Classes

Anup Rao

May 9, 2024

Max Cut

Given a graph $G = (V, E)$, a subset $S \subset V$ is called a cut of the graph. The size of the cut is the number of edges that cross from S to $V - S$. It is known to be NP-hard to compute the MAX-cut of a graph. Here we give a simple randomized algorithm that will compute a cut that is half as big as the biggest cut in expectation.

The algorithm is just to pick the subset S at random, by including every vertex in S with probability half. For each edge e, let X_e be the random variable that is 1 if e goes from S to $V - S$, and 0 otherwise. Then we see that the size of the cut is exactly $\sum_{e \in E} X_e$. We can compute $\mathbb{E}[X_e] = 1/2$, and so by linearity of expectation,

$$\mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e] = |E|/2.$$

Fingerprinting

Suppose Alice has an n-bit string x and Bob has an n-bit string y, and they want to check that they are equal. Naively this takes n bits of communication between them. We can do much better using randomization.

Alice samples a random prime number p from the set of primes that are less than $cn \ln n$, for some constant c that we shall pick later. She then sends p and $x \mod p$ to Bob. Bob checks that $x \mod p$ is equal to $y \mod p$. Thus they only need to communicate $O(\log n)$ bits in this process.

If $x = y$, this will always produce the right outcome. We shall argue that if $x \neq y$, the probability that they make a mistake is going to be very small. To do this, we need a theorem:

Theorem 1 (Prime number theorem)**.** *Let $\pi(a)$ denote the number of primes that are at most a. Then* $\lim_{a \to \infty} \frac{\pi(a)}{a/\ln a} = 1$.

When $x \neq y$, the above process fails only when p divides $x - y$. Since $|x - y| \leq 2^n$, $x - y$ can have at most n prime factors. On the other hand, by the prime number theorem, the number of primes of size up to $cn \ln n$ is at least $cn \ln n / (\ln(cn \ln n)) = \Omega(cn)$. Thus the probability that the prime Alice picks divides $x - y$ is at most $O(1/c)$.

Randomized Classes

There are several different ways to define complexity classes involving randomness. A turing machine with access to randomness is just like a normal turing machine, except it is allowed to toss a random coin in each step, and read the value of the coin that was tossed.

BPP

We say that the randomized machine computes the function f if for every input x, $\Pr_r[M(x,r) = f(x)] \geq 2/3$, where the probability is taken over the random coin tosses of the machine M. **BPP** is the set of functions that are computable by polynomial time randomized turing machines in the above sense.

RP

We shall say that $f \in$ **RP** if there is a randomized machine that always compute the correct value when $f(x) = 0$, and computes the correct value with probability at least $2/3$ when $f(x) = 1$.

ZPP

Finally, we define the class **ZPP** to be the set of boolean functions that have an algorithm that *never* makes an error, but whose *expected* running time is polynomial in n.

The true identity of **ZPP**

Theorem 2. **ZPP** $=$ **RP** $\cap$ *co***RP**.

Proof Suppose $f \in$ **ZPP**, via a randomized algorithm M whose expected running time is $t(n)$. Consider the algorithm that simulates M for $10t(n)$ steps, and outputs 0 if the simulation halts. Then clearly, the algorithm only makes an error if the correct answer is 1. On the other hand, the probability that running time of M exceeds $10t(n)$ is at most $1/10$ (or else the expected running time would exceed $t(n)$. Thus we obtain an **RP** algorithm. The same idea (reversing the roles of 0 and 1) gives a *co***RP** algorithm.

For the other direction, suppose f has an **RP** algorithm M_1 and a *co***RP** algorithm M_0. Then on input x consider the algorithm that alternatively runs $M_0(x), M_1(x), M_0(x), \ldots$ until either $M_1(x)$ outputs 1, or $M_0(x)$ outputs 0. If $M_1(x) = 1$, then it must be that $f(x) = 1$. Similarly if $M_0(x) = 0$, it must be that $f(x) = 0$. In any case, one of these two algorithms will verify the value of x in an expected constant number of runs. ■

Error reduction

It turns out the constant $2/3$ is not very important in the above definitions. That is because every probabilistic polynomial time algorithm with error $1/3$ yields a probailistic polynomial time algorithm with exponentially small error.

For example, if we have an algorithm M for f proving that $f \in \mathbf{RP}$, we can obtain a new algorithm by running $M(x)$ t times. If any of the runs outputs 1 we output 1. If all runs give an output of 0, we output 0.

We see that if $f(x) = 0$, then all runs will output 0 and our new algorithm will output 0 with probability 1. On the other hand, if $f(x) = 1$, the probability that all runs output 0 is $1/3^t$, which is exponentially small in t.