Instructor: Elad Verbin

Notes by: Zhang Zhiqiang and Yu Wei

1 Randomized complexity

So far our notion of a realistic computational device was the deterministic polynomial-time Turing Machine. However, this model does not account for the possibility to use randomness, a resource that can arguably be found in nature and used in computations. This gives the notion of a randomized Turing Machine which, in addition to its other tapes, has access to a *random tape* which is initialized with a sequence of random and uncorrelated bits $r_1, r_2, \cdots \sim \{0,1\}$.[1]

As soon as we allow the algorithm access to a random tape, its behavior is no longer completely determined by the input x. Now for every x we have a distribution of outputs depending on the how the random tape was initialized. Often it will be convenient to speak of this randomness explicitly, and we can indeed think of a randomized algorithm A as a *deterministic* algorithm that takes both a real input x and a "random" input $r = (r_1, r_2, \dots)$, denoting the setting of the random tape.

An issue that occurs in defining the *running time* of randomized computations is that the running time itself may be a random variable. It might even be that for some settings of the randomness the computation never terminates. However, much of the time we will only study computations in which the running time is bounded by $t(n)$ for all inputs of length n and all settings of the random tape. In this case we say that the randomized algorithm runs in time $t(n)$, and without loss of generality we can then model the random tape of the algorithm as a string of length $t(n)$.

For decision problems, we will think of a randomized algorithm as "solving" the problem as long as it behaves correctly on *all* inputs but for "most" choices of the randomness. Depending on what we allow the algorithm to do on the remaining, "bad" choices of randomness we obtain three different definitions of randomized complexity classes.

The first option is to require the algorithm to be correct over most choices of the randomness, but to output an arbitrary answer otherwise. This yields the class of decision problems BPP (for bounded probabilistic polynomial-time).

Definition 1. *The class* BPP *consists of all decision problems L for which there exists a randomized algorithm A running in probabilistic polynomial-time and such that for every $x \in \{0,1\}^*$,*

$$\Pr[A(x) = L(x)] \geq 2/3$$

where the probability is taken over the setting of the random tape of the algorithm A.

[1] Whether or not sufficiently many such uncorrelated and random bits can be found in nature is somewhat questionable; these questions are partially addressed in the study of randomized algorithms, specifically (but from different perspectives) in the areas of *derandomization* and *randomness extraction*.

As we will see soon, the choice of the constant 2/3 is irrelevant, as long as the constant is somewhat bigger than 1/2. Notice that if we replace the constant 2/3 with 1, we obtain exactly the class P. So in particular P $\subseteq$ BPP.

Certain randomized algorithms have a *one-sided error* property; they never accept "no" instances of the problem, but may fail on a small fraction of "yes" instances.

Definition 2. *The class* RP *consists of all decision problems L for which there exists a randomized algorithm A running in probabilistic polynomial-time and such that for every $x \in \{0,1\}^*$,*

$$\begin{aligned} x \in L &\implies \Pr[A(x) = 1] \geq 1/2 \\ x \notin L &\implies \Pr[A(x) = 0] = 1. \end{aligned}$$

Again, the choice of constant 1/2 is irrelevant, as long as this quantity is bounded away from zero. Similarly, we can have algorithms that are always correct on the "yes" instances but may err on some fraction on "no" instances. This is the class coRP. Observe that $L \in$ coRP iff $\overline{L} \in$ RP.

So far the algorithms we look at are always efficient and mostly correct. We can also consider algorithms that are mostly efficient, but always correct. Say that a randomized algorithm A runs in *expected polynomial time* if there is a polynomial p such that for all x, the expected running time of A on input x is at most $p(x)$.

Definition 3. *The class* ZPP *consists of all decision problems L that are decided by algorithms A such that for all x, $\Pr[A(x) = L(x)] = 1$ and A runs in expected polynomial time.*

Alternatively, ZPP can be defined as the class of decision problems L for which there exists a randomized algorithm B that always runs in polynomial time and on every input x outputs either $L(x)$ or the special symbol $'?'$, with the property that for every x,

$$\Pr[B(x) = L(x)] \geq 1/2.$$

Here is a sketch of the equivalence of the two definitions: To turn algorithm A into algorithm B, run $A(x)$ for at most $2p(|x|)$ steps; if A has finished within this time bound output the answer (which is guaranteed to be correct), otherwise output $'?'$. Conversely, given algorithm B, you can obtain A from B by running $B(x)$ sequentially for as many times as necessary until $B(x)$ outputs an answer different from $'?'$.

2 Amplifying the success probability

We now show that the constants that determine the success probability of randomized algorithms are indeed arbitrary. We focus on BPP algorithms. Suppose you have an algorithm for a decision problem L such that

$$\begin{aligned} x \in L &\implies \Pr[A(x) = 1] \geq c(|x|) \\ x \notin L &\implies \Pr[A(x) = 1] \leq s(|x|). \end{aligned}$$

If $c(n)$ and $s(n)$ are somewhat bounded away from one another, we can make c very close to 1 and s very close to zero by running the algorithm independently $m = \text{poly}(n)$ times and observing what fraction of copies accept. If this number is closer to $m \cdot c(n)$ than to $m \cdot s(n)$ we accept, otherwise we reject.

Let X_i be an indicator random variable for the ith run of the algorithm accepting and $X = X_1 + \cdots + X_m$, the number of accepting trials. Let $\mu = \mathrm{E}[X]$. If $x \in L$, then $\mu \geq m \cdot c(n)$, while if $x \notin L$, then $\mu \leq m \cdot c(n)$.

Recall the Chernoff bound, which tells us that if m is large, then X is very close to its expectation with extremely high probability:

Theorem 4 (Chernoff bound)**.** *Let $X_1, X_2, \cdots, X_m$ are independent random variables with $\Pr(X_i = 1) = p, \Pr(X_i = 1) = 1 - p$. Set $X = X_1 + \cdots + X_m$ and $\mu = \mathrm{E}[X]$. Then*

$$\Pr\big[|X - pm| \geq \epsilon m\big] \leq e^{-\epsilon^2 m/2p(1-p)}$$

We set $\epsilon = (c(n) - s(n))/2$ to obtain the following consequence:

$$\begin{aligned} x \in L \quad &\Longrightarrow \quad \Pr[X \leq (c(n) + s(n))m/2] \leq e^{-\delta(n)m} \\ x \notin L \quad &\Longrightarrow \quad \Pr[X \geq (c(n) + s(n))m/2] \leq e^{-\delta(n)m} \end{aligned}$$

where $\delta(n) = (c(n) - s(n))^2/2$. So as long as $c(n) - s(n) = n^{-\Omega(1)}$ – that is, $c(n)$ and $s(n)$ are not too close, we can set $m = p(n)/\delta(n)$, where p is an arbitrary polynomial. Then the algorithm that runs m independent copies of A and accepts whenever $(c(n) + s(n))m/2$ of those copies accepts has failure probability at most $2^{-p(n)}$.

3 Relations between randomized complexity classes

P $\subseteq$ ZPP**:** Any algorithm that runs in polynomial time runs in expected polynomial time.

ZPP $\subseteq$ RP $\cap$ coRP**:** We show ZPP $\subseteq$ RP. Consider the definition of ZPP where an algorithm B returns either the correct answer or $'?'$. When B outputs 0 or 1, output this answer; when B outputs $'?'$, output 0.

RP $\cap$ coRP $\subseteq$ ZPP**:** Given an RP algorithm A and a coRP algorithm B for the same problem, on input x, run both A and B. If $A(x)$ outputs 0, output 0; if $B(x)$ outputs 1, output 1; otherwise, output $'?'$.

RP $\subseteq$ BPP**:** By our discussion in the previous section, the probability 1/2 in the definition of RP can be replaced by 2/3.

RP $\subseteq$ NP**:** Think of the randomness r used by the algorithm as an NP witness. If x is a "no" instance, then $A(x, r)$ rejects for all r. If x is a "yes" instance, then $A(x, r)$ accepts with nonzero probability, so there is some r for which $A(x, r)$ acccepts.

We now state some containments of BPP.

4 Adleman's Theorem

Theorem 5 (Adelman)**.** BPP $\subseteq$ P/poly.

Proof. Let $L \in$ BPP. By our discussion on amplifying the success probability, L has a BPP type algorithm such that for every x,

$$\Pr_r[A(x,r) = L(x)] \geq 1 - 1/2^{|x|+1}.$$

We fix the input size $|x| = n$ and show that the same string r_n can be good for all inputs of size n simultaneously.

$$\begin{aligned} \Pr_{r_n}[\forall x \in \{0,1\}^n, A(x,r_n) = L(x)] &= 1 - \Pr_{r_n}[\exists x \in \{0,1\}^n, A(x,r_n) \neq L(x)] \\ &\geq 1 - \sum_{x \in \{0,1\}^n} \Pr_{r_n}[A(x,r_n) \neq L(x)] \\ &\geq 1 - 2^n \cdot 2^{-n-1} = 1/2, \end{aligned}$$

so there is a r_n of length poly(n) that behaves correctly on all length n inputs. Let circuit C_n simulate $A(x, r_n)$ on input x when $|x| = n$. Then C_n computes L, so $L \in$ P/poly. □

This result indicates that randomness is not too powerful. In particular it is unlikely that NP $\subseteq$ BPP, for in that case we would have that NP $\subseteq$ P/poly and by the Karp-Lipton-Sipser theorem, $\Sigma_2 = \Pi_2$.

5 Randomized computation and the polynomial hierarchy

The result BPP $\subseteq$ P/poly seems to indicate that randomness does not add much to the power of deterministic computation. So it seems reasonable to conjecture that BPP = P. In a few lectures we will provide some evidence for this claim. However we cannot even prove that BPP $\subseteq$ NP. The best statement of this kind we can prove is the following:

Theorem 6 (Gacs and Sipser)**.** BPP $\subseteq \Sigma_2$.

We show a proof of this statement by Lautemann.

Proof. Let $L \in$ BPP and let $A(x,r)$ be a BPP type algorithm for L with error probability $\leq 1/3m$ that uses $m = m(n)$ bits of randomness on inputs of length n.

To show that $L \in \Sigma_2$, we show the following characterization of L:

$$x \in L \quad \Leftrightarrow \quad \exists y_1, \cdots, y_m \in \{0,1\}^m \; \forall z \in \{0,1\}^m : \bigvee_{i=1}^{m} A(x, y_i \oplus z) = 1.$$

This statement immediately yields a Σ_2-type algorithm for L.

First, suppose $x \in L$. When we choose $y_1, y_2, \cdots, y_m$ independently uniformly randomly,

$$\begin{aligned} \Pr_{y_1,\cdots,y_m}[\exists z : A(x, y_i \oplus z) = 0 \text{ for all } i] &\leq 2^m \Pr_{y_1,\cdots,y_m}[A(x, y_i) = 0 \text{ for all } i] \\ &\leq 2^m \cdot (1/3m)^m < 1 \end{aligned}$$

So there must exist $y_1, \cdots, y_m$ such that for some z, $A(x, y_i \oplus z) = 1$.

Now, suppose $x \notin L$. We now need to prove that

$$\forall y_1, \cdots, y_m \in \{0,1\}^m \ \exists z \in \{0,1\}^m : \bigwedge_{i=1}^{m} A(x, y_i \oplus z) = 0.$$

Fix an arbitrary choice for $y_1, \cdots, y_m$. Then for a random $z \sim \{0,1\}^m$,

$$\begin{aligned} \Pr_z[A(x, y_i \oplus z) = 1 \text{ for some } i] &\leq \sum_{i=1}^{m} \Pr_z[A(x, y_i \oplus z) = 1] \\ &\leq m \cdot (1/3m) = 1/3 < 1 \end{aligned}$$

so we can always choose z such that $A(x, y_i \oplus z) = 0$ for all i. □