1 Randomized Complexity

1.1 Randomness is powerful

We are interested in knowing: to what extent does randomness give computational power and what are its limitations.

Randomness is essential in some computational tasks, for example cryptography. However, it is not clear if it helps in computation (we believe the answer is no). In the past history, people were able to obtain deterministic algorithm by first designing randomized algorithm and later eliminating the randomness in them. This is called derandomization. For some type of randomized algorithms, people found methods to derandomize them. It is conjectured that randomness doesn’t give extra power to computation. It’s only a easy way to design algorithms and have smaller time complexity. For example, PRIMALITY. To build succinct data-structure, randomness needs to be removed because randomness cannot to compressed.

Suppose a language $L \subset \{0, 1\}^*$. What’s a good reason for a randomized algorithm to decide a problem? Say, 3-COLORING. There are two natural notions:

- Only works for most graphs.
- Solve the problem for all inputs with high probability.

It is not clear what is a good distribution for random graphs one sees in practice, and in most cases it is domain specific. We will focus on the second definition, wish requires to solve the problem on all inputs.

**Definition 1 (Randomized Turing machine and its language)** A randomized Turing machine $M$ decides $L$ if

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq \frac{2}{3}$$

$$x \notin L \Rightarrow \Pr[M(x) = 0] \geq \frac{2}{3}$$

**Definition 2 (BPP)** $\mathsf{BPP} \overset{\text{def}}{=} \text{“languages decidable by randomized poly-time computation.”}$

Another equivalent model which will be useful sometimes is one where the randomness is explicit. Let $M(x, r)$ be a deterministic Turing machine where $x$ is the input and $r$ is the randomness to be used, where we assume $|r| = m \leq \text{poly}(|x|)$. Then, $M$ decides a language $L$ if

$$\forall x, \Pr_{r \in \{0, 1\}^m}[M(x, r) = L(x)] \geq \frac{2}{3}.$$

1.2 Error reduction

**Definition 3 (BPP_p)** $\mathsf{BPP}_p \overset{\text{def}}{=} \text{“languages for which exists randomized poly-time machine } \Pr[M(x) = L(x)] \geq p \text{”}$ where $p > \frac{1}{2}$.

For example, we defined $\mathsf{BPP} = \mathsf{BPP}_{\frac{2}{3}}$. We will next show that error by can reduced by repeating the computation a few times, and taking the majority value.
Theorem 4 \( \text{BPP} = \text{BPP}_{\frac{1}{2} + \frac{1}{nc}} = \text{BPP}_{1 - 2^{-nc}} \quad \forall c > 0 \)

We will use the Chernoff-Hoeffding Bound.

Theorem 5 (Chernoff-Hoeffding Bound) Let \( X_1, \cdots, X_n \in \{0, 1\} \) be random variables, independent. Suppose \( \mathbb{E}[X_i] = p \) for any \( i \in [n] \). Then
\[
\Pr\left[ \left| \frac{1}{n} \sum_{i=1}^{n} X_i - p \right| > \epsilon \right] \leq 2e^{-2\epsilon^2 n}
\]

Proof of Theorem 4:

Let \( M \) be a poly-time TM s.t. \( \forall x, \Pr_r \left[ M(x, r) = L(x) \right] > \frac{1}{2} + \frac{1}{nc} \). Let \( k \) to be chosen later and define \( M'(x, (r_1, \cdots, r_k)) = \text{MAJ}(M(x, r_1), \cdots, M(x, r_k)) \). Fix input \( x \), define random variable
\[
X_i \overset{\text{def}}{=} \begin{cases} 1 & M(x, r_i) = L(x) \\ 0 & \text{If not} \end{cases}
\]

Let \( p \overset{\text{def}}{=} \mathbb{E}[X_i] > \frac{1}{2} + \frac{1}{nc} \). Then the probability \( M'(x, (r_1, \cdots, r_k)) \neq L(x) \) is bounded by
\[
\Pr[M'(x, (r_1, \cdots, r_k)) \neq L(x)] = \Pr\left[ \frac{1}{k} \sum_{i=1}^{k} X_i < \frac{1}{2} \right]
\]
\[
= \Pr\left[ \frac{1}{k} \sum_{i=1}^{k} X_i - p < -\frac{1}{nc} \right]
\]
\[
\leq 2e^{-2 \cdot \frac{1}{nc} \cdot k} \quad \text{Chernoff bound, } k = 2n^{3c}
\]
\[
= O(2^{-n^c}).
\]

Next time: examples of randomized algorithms.