Lecture 7

1 Introduction to Randomized Algorithms

A randomized algorithm is an algorithm which involves randomness. Such algorithms are sometimes simpler or more efficient than their deterministic counterparts. We start by describing and analyzing two fairly simple examples.

1.1 Verifying Polynomial Identities

Consider the following natural problem. Given two polynomials $F(x) = \prod_{i=1}^{d}(x - a_i)$ and $G(x) = x^d + \sum_{j=0}^{d-1} b_j x^j$, we would like to know whether $F(x) \equiv G(x)$ or not. A very straightforward way of doing this is by transforming $F(x)$ to its canonical form by multiplying its terms and then comparing coefficients. This approach requires $\Theta(d^2)$ multiplications and additions. The following simple randomized algorithm is significantly faster.

Randomized Algorithm for Comparing Polynomials:

Input: Polynomials $F(x) = \prod_{i=1}^{d}(x - a_i)$ and $G(x) = x^d + \sum_{j=0}^{d-1} b_j x^j$.
Output: $F(x) \equiv G(x)$ or $F(x) \not\equiv G(x)$.

1. Choose an integer $r \in \{1, 2, \ldots, 100d\}$ uniformly at random.
2. If $F(r) \neq G(r)$, then output $F(x) \not\equiv G(x)$, otherwise output $F(x) \equiv G(x)$.

Let us analyze the correctness and complexity of this algorithm. Assume, for now, that choosing $r \in \{1, 2, \ldots, 100d\}$ uniformly at random takes constant time. Calculating $F(r)$ and $G(r)$ takes $\Theta(d)$ time. Hence, the running time of this algorithm is $\Theta(d)$.

Now, assume first that $F(x) \equiv G(x)$. In this case, clearly $F(r) = G(r)$ and thus the algorithm will output the correct answer. Assume then that $F(x) \not\equiv G(x)$. If $F(r) \neq G(r)$, then again the algorithm will output the correct answer. On the other hand, if $F(r) = G(r)$, then the algorithm will output $F(x) \equiv G(x)$ which is incorrect. Let us estimate the probability of this event. If $F(x) \not\equiv G(x)$, then $F(x) - G(x)$ is not the zero polynomial and thus has at most d roots. In particular, $F(x) - G(x)$ has at most d roots in the set $\{1, 2, \ldots, 100d\}$. Since r was sampled from this set uniformly, it follows that $\mathbb{P}(F(r) = G(r)) \leq 1/100$. To summarize, whenever the algorithm outputs $F(x) \not\equiv G(x)$, this is the correct answer. On

the other hand, whenever the algorithm outputs $F(x) \equiv G(x)$, this is the correct answer with probability at least 0.99. This algorithm has two advantages and one disadvantage. On the one hand, it is extremely simple and very fast ($\Theta(d)$ compared to $\Theta(d^2)$ for the trivial solution). On the other hand, it might output an incorrect answer. This disadvantage might be easier to bare, if we could make the error probability much smaller than 1/100 (say, smaller than the probability of dying from a lightning strike). This can be done by enlarging the size of the set from which r is sampled. However, this may effect the complexity of the algorithm (the complexity of sampling from a set is in fact affected by the size of the set). Moreover, working with very large numbers may prove problematic for computers. Here is a better (and more general) solution.

An Improved Randomized Algorithm for Comparing Polynomials:

Input: Polynomials $F(x) = \prod_{i=1}^{d}(x - a_i)$ and $G(x) = x^d + \sum_{j=0}^{d-1} b_j x^j$, and a positive integer k.
Output: $F(x) \equiv G(x)$ or $F(x) \not\equiv G(x)$.

1. For $1 \leq i \leq k$
   (a) Choose an integer $r_i \in \{1, 2, \ldots, 2d\}$ uniformly at random.
   (b) If $F(r_i) \neq G(r_i)$, then stop and output $F(x) \not\equiv G(x)$.
2. Output $F(x) \equiv G(x)$.

A similar analysis to the one we performed above shows that the complexity of this algorithm is $\Theta(kd)$ (which is $\Theta(d)$ if k is a constant). Moreover, whenever the algorithm outputs $F(x) \not\equiv G(x)$, this is the correct answer, and whenever the algorithm outputs $F(x) \equiv G(x)$, this is the correct answer with probability at least $1 - 2^{-k}$.

1.2 A Randomized Min-Cut Algorithm

Let $G = (V, E)$ be a connected graph. An *edge-cut* of G is a set $A \subseteq E$ such that $G \setminus A$ is disconnected. Finding an edge-cut of minimum size is a classical problem in computer science (for which there are efficient deterministic algorithms). We present here a very simple randomized algorithm. It is based on the operation of *edge contraction*. Given a multigraph G without loops and an edge $uv \in E(G)$, contracting the edge uv is done by merging the vertices u and v into a new vertex z_{uv}, namely, by deleting all edges connecting u and v and replacing any edge $xy \in E(G) \cap (\{u, v\} \times (V(G) \setminus \{u, v\}))$ with the edge $z_{uv}y$. Observe that the resulting multigraph, which we denote by G/uv, may contain parallel edges but not

loops.

A randomized algorithm for finding a minimum cut:

Input: A connected graph G on n vertices.
Output: An edge-cut of G.

1. Let $G_0 = G$.
2. For $1 \leq i \leq n-2$
   (a) Pick an edge $e_i \in E(G_{i-1})$ uniformly at random.
   (b) Set $G_i = G_{i-1}/e_i$.
3. Output $E(G_{n-2})$.

Since every step in the for loop takes $O(n)$ time, it readily follows that the time complexity of this algorithm is $O(n^2)$. Our main goal is to prove the following result.

Theorem 1.1. *The algorithm outputs a min-cut with probability at least* $\binom{n}{2}^{-1}$.

Proof. Let $A \subseteq E(G)$ be some min-cut of G; denote its size by k. For every $1 \leq i \leq n-2$ let E_i denote the event "$e_i \notin A$". Note that

$$\mathbb{P}\left(\text{the algorithm outputs a min-cut}\right) \geq \mathbb{P}\left(\text{the algorithm outputs } A\right) \geq \mathbb{P}\left(\bigcap_{j=1}^{n-2} E_j\right).$$

Indeed, since A is an edge-cut of G, removing it splits $V(G)$ into two sets S and $V(G)\setminus S$ with no edges connecting them. If we contract edges with both endpoints in S or in $V(G)\setminus S$ for $n-2$ times, then both S and $V(G)\setminus S$ turn into single vertices and the edges connecting these vertices are precisely the edges of A. It thus remains to prove that $\mathbb{P}\left(\bigcap_{j=1}^{n-2} E_j\right) \geq \binom{n}{2}^{-1}$. Since A is a min-cut and its size is k, every cut in G has size at least k. In particular, the minimum degree is at least k and thus $|E(G)| \geq kn/2$. Since e_1 is chosen uniformly at random from $E(G)$, it follows that

$$\mathbb{P}\left(E_1\right) = \frac{|E(G)\setminus A|}{|E(G)|} \geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n}.$$

Observe that $|V(G_1)| = n-1$ and that the size of a min-cut in G_1 is at least k (as any edge-cut of G_1 is also an edge-cut of G). Therefore,

$$\mathbb{P}\left(E_2|E_1\right) \geq 1 - \frac{k}{k(n-1)/2} = 1 - \frac{2}{n-1}.$$

An analogous argument implies that

$$\mathbb{P}\left(E_i | \bigcap_{j=1}^{i-1} E_j\right) \geq 1 - \frac{k}{k(n-i+1)/2} = 1 - \frac{2}{n-i+1}$$

holds for every $3 \leq i \leq n-2$. We conclude that

$$\begin{aligned}
\mathbb{P}\left(\bigcap_{j=1}^{n-2} E_j\right) &= \mathbb{P}\left(E_1\right) \cdot \mathbb{P}\left(E_2 | E_1\right) \cdot \ldots \cdot \mathbb{P}\left(E_{n-2} | \bigcap_{j=1}^{n-3} E_j\right) \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) \\
&= \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right) = \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right) \cdot \ldots \cdot \left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)}.
\end{aligned}$$

□

As in the algorithm for verifying polynomial identities, we can make the error probability arbitrarily small by running the algorithm many times and returning as output the smallest cut found. For example, if we run this algorithm $n(n-1)\ln n$ times, then the probability the algorithm fails to output a min-cut is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)\ln n} \leq e^{-2\ln n} = \frac{1}{n^2}.$$