CSE200: Computability and complexity
Randomized algorithms

Shachar Lovett
May 17, 2013

1 Randomized algorithms

Randomness is a resource which can be use computationally.

**Definition 1.1.** A randomized Turing machine is a Turing machine which allows to "toss coins" during the computation. Formally, we can use either of the following equivalent definition.

- Implicit: the Turing machine has two transition functions, and it chooses in each step one of them randomly and uniformly.
- Explicit: the Turing machine has an input tape, and a random tape, filled with random bits. In the random tape the head can only move to the right (e.g. it is read once).

**Definition 1.2 (BPTIME).** The class $\text{BPTIME}(T(n))$ is the class of all languages $L \subseteq \{0,1\}^*$ for which there exists a randomized Turing machine $M$ such that $M$ always halts in at most $T(n)$ steps and

\[
\begin{align*}
  x \in L &\Rightarrow \Pr[M(x) = 1] \geq 2/3, \\
  x \notin L &\Rightarrow \Pr[M(x) = 1] \leq 1/3.
\end{align*}
\]

We can write this more succinctly as

\[
\Pr[M(x) = L(x)] \geq 2/3.
\]

**Definition 1.3 (BPP).** $BPP = \bigcup_{c \geq 1} \text{BPTIME}(n^c)$ is the class of languages solvable by poly-time randomized algorithms. Equivalently, $L \in BPP$ if there exists a deterministic poly-time Turing machine $M(x, r)$ with $|r| \leq |x|^c$ and

\[
\begin{align*}
  x \in L &\Rightarrow \Pr_r[M(x, r) = 1] \geq 2/3, \\
  x \notin L &\Rightarrow \Pr_r[M(x, r) = 1] \leq 1/3.
\end{align*}
\]
2 Error reduction

The error bounds $2/3, 1/3$ in the definition of $BPP$ are arbitrary. We can reduce the error by repetition and taking majority. For $p > 1/2$ let $BPP_p$ be defined as follows: $L \in BPP_p$ if there exists a poly-time Turing machine $M(x, r)$ for which

$$\Pr_r[M(x, r) = L(x)] \geq p.$$ 

Clearly, if $1/2 < p < q$ then $BPP_q \subset BPP_p$.

Lemma 2.1. For any $c > 0$,

$$BPP_{1/2 + n^{-c}} = BPP_{1 - 2^{-n^c}} = BPP.$$ 

Proof. The idea is simple: repeat the computation a few times, and take majority. The bounds follow by the limitations of repeating the computation a polynomial number of times. Let $p = 1/2 + 2^{-n^c}$ and let $L \in BPP_p$. That is, there exists a poly-time Turing machine $M(x, r)$ for which

$$\Pr_r[M(x, r) = L(x)] \geq p.$$ 

Let $k = n^{O(1)}$ be odd and $r' = r_1, \ldots, r_k$ with each $r_i$ the same length as $r$. Define $M'(x, r')$ as

$$M'(x, r') = MAJORITY(M(x, r_1), \ldots, M(x, r_k)).$$

Clearly $M'$ is a poly-time machine. We need to compute $\Pr_r[M(x, r') = L(x)]$. We will use the Chernoff-Hoeffding bound.

Theorem 2.2 (Chernoff-Hoeffding). Let $X_1, \ldots, X_k$ be independent random variables with $X_i \in \{0, 1\}$ and $\mathbb{E}[X_i] \geq p$. Then

$$\Pr\left[\sum_{i=1}^{k} X_i - pk \geq \varepsilon k\right] \leq \exp(-2\varepsilon^2 k).$$

and

$$\Pr\left[\sum_{i=1}^{k} X_i - pk \leq -\varepsilon k\right] \leq \exp(-2\varepsilon^2 k).$$

In our case, let $X_i = 1_{M(x, r_i) = L(x)}$. Then $\mathbb{E}[X_i] \geq p$. $M'(x, r') = L(x)$ if the majority of $M(x, r_i)$ gave the correct answer; that is if $\sum_{i=1}^{k} X_i \geq k/2$. By the Chernoff-Hoeffding bound, the probability this does not hold is

$$\Pr[\sum_{i=1}^{k} X_i \leq k/2] = \Pr[\sum_{i=1}^{k} X_i - pk \leq -(p - 1/2)k] \leq \exp(-2(p - 1/2)^2 k).$$

Recall that $p = 1/2 + n^{-c}$. Hence setting $k = n^{2c}$ gives that

$$\Pr_{r'}[M'(x, r') = L(x)] \geq 1 - \exp(-n^{-c}).$$
3 Examples of randomized algorithms

3.1 Finding the median

Let $A = \{a_1, \ldots, a_n\}$ be a multi-set. We wish to find their median. More generally, we want to find the $k$-th smallest element, where median corresponds to $k = n/2$. Define:

$$\text{ELEMENT}(k, A):$$

1. Choose randomly $p \in [n]$ (a pivot).
2. Let $L = \{i : a_i \leq a_p\}$ and $H = \{i : a_i > a_p\}$ be the set of low or high elements relative to the pivot $a_p$, respectively.
3. If $|L| \geq k$ then return $\text{ELEMENT}(k, L)$,
4. If $|L| < k$ then return $\text{ELEMENT}(k - |L|, H)$.

We claim that on average, $\text{ELEMENT}(k, A)$ runs in time $O(|A|)$.

**Theorem 3.1.** Let $T(k, A)$ be the expected running time the algorithm takes on inputs $k, A$. Let $T(n)$ denote the maximum of $T(k, A)$ for $|A| = n$ and $1 \leq k \leq n$. Then $T(n) = O(n)$.

**Proof.** Each run of the algorithm (without the recursion) takes linear time, say $cn$. We will prove (by induction) that $T(n) \leq 10cn$. Let $\ell = |L|$ be a random variable. Note that $\ell$ is uniform in $\{1, \ldots, n\}$. We can bound

$$T(k, A) \leq cn + \frac{1}{n} \sum_{\ell=1}^{n} \max(T(\ell), T(n - \ell)) \leq cn + \frac{2}{n} \sum_{\ell=n/2+1}^{n} 10\ell.$$

Now,

$$\sum_{\ell=n/2+1}^{n} \ell = \frac{n(n+1)}{2} - \frac{n/2(n/2 + 1)}{2} \leq \frac{3n^2 + 4n}{8}$$

Hence

$$T(k, A) \leq cn + 10c((3/4)n + 1) = 8.5cn + 10c \leq 10cn$$

for $n \geq 10$. 

3.2 Primality testing

Let $n \in \mathbb{N}$. We want to test if $n$ is prime or not. We will use Fermat small theorem: if $p$ is prime then $a^{p-1} = 1 \pmod{p}$ for all $1 \leq a \leq p - 1$. Formally, this is not an if-and-only-if criteria. The Rabin-Miller algorithm is a randomized algorithm which uses this to check if a number is prime or not.
Miller-Rabin Primality test (n):
1. Choose randomly $1 \leq a \leq n - 1$.
2. If $a^{n-1} \not\equiv 1 \pmod{n}$ return NO.
3. Let $r = n - 1$.
4. While $a^r = 1 \pmod{n}$ and $r$ is even, set $r := r/2$.
5. If $a^r = 1 \pmod{n}$ or $a^r = -1 \pmod{n}$ then return YES, else return NO.

If $n$ is prime the algorithm always return YES because the only square root of 1 modulo a prime is \(-1\). If $n$ is not prime then it can be shown that 1 has at least four square roots, and that with probability at least $3/4$ a random $a$ will be a witness to the non-primality of $n$. This can be amplified by repeating the test a few times.

### 3.3 Undirected connectivity in graphs

Let $G$ be an undirected graph on $n$ vertices. A random walk on $G$ is a process that at every step moves from a vertex to an adjacent vertex. To check if $s,t$ are in the same connected component, run the following randomized algorithm:

1. Init $v := s$.
2. Loop for at most $O(n^3)$ steps:
   (a) Move $v$ to a random neighbor of $v$.
   (b) If $v = t$ return YES.
3. Return NO.

**Theorem 3.2.** For any vertex $s$ of $G$, a random walk of length $O(n^3)$ will, with high probability, visit all vertices in the connected component of $G$. In particular, it will detect whether $t$ is reachable from $s$.

To see this will not work on directed graphs, consider the following (strongly connected) graph: $V = \{1, 2, \ldots, n\}$ and edges $(i, i+1), (i, 1)$ for all $i = 1, \ldots, n$. If we start the walk at 1 it will take on average $2^n$ steps to reach $n$.

### 3.4 Schöning randomized 3SAT algorithm

Let $\phi(x_1, \ldots, x_n) = C_1(x) \land \ldots \land C_m(x)$ be a 3CNF formula. A trivial algorithm to check whether $\phi$ is satisfiable runs in $2^n$ time. The following randomized algorithm of Schöning runs in time $\approx (4/3)^n$. 


3SAT-RandomWalk:

1. Choose $a \in \{0, 1\}^n$ randomly.

2. Repeat for at most $3n$ steps:
   
   (a) If $\phi(a) = 1$ return $a$.
   
   (b) Otherwise, let $C_i$ be a clause on which $C_i(a) = 0$. Let $x_j$ be a random variable in $C_i$.
   
   (c) Set $a_j = 1 - a_j$.

**Theorem 3.3.** If $\phi$ is satisfiable, then with probability at least $(3/4)^n$ RandomWalk finds a satisfiable solution.

Hence, if after $\approx (4/3)^n$ runs of RandomWalk no solution was found, then with high probability $\phi$ is unsatisfiable.

3.5 Finding simple paths in graphs

Let $G = (V, E)$ be an undirected graph. The problem is to decide whether $G$ has a simple path of length $k$. A simple solution takes $n^k$ enumerating all the possible solutions. We will show a randomized algorithm which finds (whp) such a coloring in time $n^{O(1)} 2^k$.

**Definition 3.4.** A random coloring $\chi : V \to [c]$ is a coloring where the color of each vertex $v$ is chosen uniformly and independently.

**Claim 3.5.** Let $\chi : V \to [k]$ be a random coloring. Let $P$ be a simple path in $G$ of length $k$. Then with probability at least $2^{-O(k)}$ the path $P$ takes all $k$ colors.

**Proof.** The probability that $P$ takes all $k$ colors is $\frac{k!}{k^k} = \left(\frac{c}{\sqrt{k}}\right) e^{-k}$. \hfill \Box

**Claim 3.6.** Let $G = (V, E)$ be a graph and $\chi : V \to [k]$ be a coloring. Then one can detect if $G$ has a path of length $k$ with all $k$ different colors can be done in time $n^{O(1)} 2^k$.

**Proof.** For a subset $A \subset [k]$ of size $|A| = a$, define

$$S_A = \{v \in V : \text{There exists a path } v = v_1, v_2, \ldots, v_k \text{ where } \{\chi(v_1), \ldots, \chi(v_k)\} = A\}.$$

We will compute $S_A$ by a dynamic program. If $|A| = 1$, e.g. $A = \{a\}$ then $S_{\{a\}} = \{v \in V : \chi(v) = a\}$ can be computed in time $n$. If $|A| > 1$ then

$$S_A = \bigcup_{a \in A} \{v \in S_{A\setminus\{a\}} : \exists u, \chi(u) = a, (u, v) \in E\}.$$

Note that we can compute $S_A$ given all $S_{A\setminus\{a\}}$ in time $O(n^2 k)$. Hence, we can compute $S_{[k]}$ in time $n^{O(1)} 2^k$. \hfill \Box

5
3.6 Polynomial identity testing

Let $p(x_1, \ldots, x_n)$ be a real polynomial given in some succinct form (for example, as the determinant of a matrix with polynomial entries). We want to check if $p \equiv 0$.

**Lemma 3.7 (Schartz-Zippel).** Let $p(x_1, \ldots, x_n)$ be a nonzero polynomial of degree $d$. Let $S \subset \mathbb{R}$ be any set of $|S| > d$. Then, if $a_1, \ldots, a_n \in S$ are chosen uniformly and independently, 

$$\Pr[p(a_1, \ldots, a_n) \neq 0] \geq 1 - \frac{d}{|S|}.$$ 

This gives a randomized algorithm to check if a low-degree polynomial is zero or not - evaluate it on a random input, say by taking $S = \{1, 2, \ldots, d/\epsilon\}$ be get an algorithm which finds (randomly) a witness for being nonzero with probability $1 - \epsilon$.

As an application, let’s apply this idea to checking if a bipartite graph has a perfect matching. Let $G = (V_1, V_2, E)$ be a bi-partite graph with $|V_1| = |V_2| = n$. A perfect matching is a permutation $\pi \in S_n$ such that $(i, \pi(i)) \in E$. In order to find if $G$ has a perfect matching, consider the following setup. Let $M$ be an $n \times n$ variables, with $M_{i,j} = x_{i,j}$ if $(i,j) \in E$ and $M_{i,j} = 0$ otherwise, where the $x_{i,j}$ are variables. Then note that

$$\det(M) = \sum_{\pi \in S_n} (-1)^{\text{sign}(\pi)} \prod_{i=1}^n M_{i,\pi(i)}$$

$$= \sum_{\pi \text{ is a perfect matching in } G} (-1)^{\text{sign}(\pi)} \prod_{i=1}^n x_{i,\pi(i)}.$$ 

The polynomial $p(x_1,1, \ldots, x_{n,n})$ in $n^2$ variables is zero iff $G$ has no perfect matchings. Hence, using the Schartz-Zippel algorithm we can randomly test if $G$ has a perfect matching.

4 One-sided and zero-sided error algorithms

A special case of randomized algorithms are algorithms which make error only on ”one side”.

**Definition 4.1 (RP).** A language $L \subset \{0,1\}^*$ is in RP if there exists a poly-time Turing machine $M(x, r)$ such that

$$x \in L \Rightarrow \Pr_r[M(x, r) = 1] \geq 2/3$$

$$x \notin L \Rightarrow \Pr_r[M(x, r) = 1] = 0$$

For example, the Miller-Rabin algorithm makes no error when $n$ is prime, and makes a bounded error when $n$ is composite. We can also define $L \in \text{coRP}$ if $L^c \in \text{RP}$. 


Definition 4.2 (ZPP). A language $L \subset \{0,1\}^*$ is in RP if there exists a randomized Turing machine $M(x)$ such that

1. For all $x \in \{0,1\}^*$ we have $M(x) = L(x)$ with probability one.
2. The expected running time of $M(x)$ is polynomial in $|x|$.

Theorem 4.3. $\text{ZPP} = \text{RP} \cap \text{coRP}$.

Proof. Let $L \in \text{RP} \cap \text{coRP}$. This means there are randomized poly-time Turing machines $M_1, M_2$ such that

$$x \in L \Rightarrow \Pr[M_1(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[M_1(x) = 1] = 0$$

and

$$x \in L \Rightarrow \Pr[M_2(x) = 1] = 0$$
$$x \notin L \Rightarrow \Pr[M_2(x) = 1] = 1$$

Consider the following machine $M(x)$ defined as follows:

1. If $M_1(x) = 1$ return "$x \in L$".
2. If $M_2(x) = 1$ return "$x \notin L$".
3. Return to step 1.

We first argue that when $M(x)$ returns an answer it must be correct. This is because $M_1(x) = 1$ can only hold for $x \in L$; and $M_2(x) = 1$ can only hold for $x \notin L$. Moreover, if the running time of $M_1(x)$ is $T_1(x)$ and of $M_2(x)$ is $T_2(x)$ then the expected running time of $M(x)$ is $O(T_1(n) + T_2(n))$. To see that, let $x \in L$. Then since $\Pr[M_1(x) = 1] \geq 2/3$, then $M(x)$ will halt in the first round, similarly if $x \notin L$. So, if we let $T(x)$ denote the expected running time of $M(x)$ we get that

$$T(x) = (2/3)(T_1(x) + T_2(x)) + (1/3)T_0(x).$$

This solves to $T(x) = O(T_1(x) + T_2(x))$ which by assumption is polynomial in $|x|$.

For the other direction, let $L \in \text{ZPP}$. There is a randomized Turing machine $M(x)$ such that $M(x) = L(x)$ with probability one. Let $T(x)$ be the random variable measuring the time that $M(x)$ runs, where we know that $\mathbb{E}[T(x)] = T_0(x) = \text{poly}(|x|)$. By Markov’s inequality,

$$\Pr[T(x) \geq 3T_0(x)] \leq 1/3.$$ 

Consider the following machine $M'(x)$:
1. run $M(x)$ for at most $3T_0(x)$ steps.

2. If $M(x)$ returns a value then return it.

3. If not, return 0.

Now, if $x \in L$ then $\Pr[M'(x) = 1] \geq 2/3$, and if $x \notin L$ then $\Pr[M'(x) = 1] = 0$. Hence $L \in RP$. If we replace ”return 0” with ”return 1” in the last step we will get a coRP machine.

5 Space bounded randomized algorithms

**Definition 5.1.** A language $L \subset \{0,1\}^*$ is in $BPL$ if it can be decided by a two-sided randomized algorithm running in logspace. It is in $RL$ if it can be decided by a one-sided randomized algorithm running in logspace.

As we already mentioned, Reingold gave a de-randomization of this algorithm (which is an RL algorithm). We suspect that $BPL = RL = L$. The best known result is

**Theorem 5.2** (Saks-Zhou). $BPL \subset SPACE(\log^{3/2} n)$.

6 Randomized algorithms vs other models

**Theorem 6.1** (Adleman’78). $BPP \subset P/poly$.

*Proof.* Let $L \in BPP$. Apply error reduction to get a poly-time Turing machine $M(x,r)$ such that for any input $x$ with $|x| = n$,

$$\Pr_r[M(x,r) = L(x)] \geq 1 - 2^{-2n}.$$ 

This means that

$$\Pr_r[\exists x \in \{0,1\}^n, M(x,r) \neq L(x)] \leq 2^n \cdot 2^{-2n} = 2^{-n}.$$ 

In particular, there must exist $r$ for which $M(x,r) = L(x)$ for all $x \in \{0,1\}^n$. The circuit will compute $M(x,r)$ with $r$ hard coded. 


**Theorem 6.2** (Sipser-Gács). $BPP \subset \Sigma_2 \cap \Pi_2$.

*Proof.* We will prove $BPP \subset \Sigma_2$. The claim $BPP \subset \Pi_2$ will follow since $BPP$ is close to negation.

Let $L \in BPP$. Apply error reduction to get a poly-time Turing machine $M(x,r)$ such that for any input $x$ with $|x| = n$, a

$$x \in L \Rightarrow \Pr_r[M(x,r) = 1] \geq 1 - 2^{-n}$$

$$x \notin L \Rightarrow \Pr_r[M(x,r) = 1] \leq 2^{-n}$$

8
We will use quantifiers to distinguish between the huge set of witnesses to \( x \in L \) and the tiny set of witnesses for \( x \notin L \). Let \( r \in \{0,1\}^m \) with \( m = \text{poly}(n) \). For a set \( R \subset \{0,1\}^m \) and \( u \in \{0,1\}^m \) let \( R + u = \{ r + u : r \in R \} \) be the shift of \( R \) by \( u \) (where the sum is modulo 2). We will need the following claims. Let \( k = 2m/n \).

**Claim 6.3.** Let \( R \subset \{0,1\}^m \) with \( |R| \leq 2^{m-n} \). For any \( u_1, \ldots, u_k \in \{0,1\}^m \), we have \( \bigcup_{i=1}^k (R + u_i) \neq \{0,1\}^m \).

**Proof.** We have \( |R + u_i| = |R| = 2^{m-n} \). Hence \( |\bigcup_{i=1}^k (R + u_i)| \leq 2^{m-n} k < 2^m \) since \( 2^{-n} k = 2^{-n} \text{poly}(n) \ll 1 \).

**Claim 6.4.** Let \( R \subset \{0,1\}^m \) with \( |R| \geq 2^{m} (1 - 2^{-n}) \). Then there exist \( u_1, \ldots, u_k \in \{0,1\}^m \) such that \( \bigcup_{i=1}^k (R + u_i) = \{0,1\}^m \).

**Proof.** The proof is by the probabilistic method. We will argue that by choosing \( u_1, \ldots, u_k \in \{0,1\}^m \) randomly then with high probability \( \bigcup_{i=1}^k (R + u_i) = \{0,1\}^m \). Hence such a choice must exist. To calculate the probability, note that for any specific \( r \in \{0,1\}^m \),

\[
\Pr_{u_1, \ldots, u_k \in \{0,1\}^m} \left[ r \notin \bigcup_{i=1}^k (R + u_i) \right] = \prod_{i=1}^k \Pr_{u_i \in \{0,1\}^m} \left[ r \notin R + u_i \right] = \prod_{i=1}^k \Pr_{u_i \in \{0,1\}^m} \left[ u_i \notin R + r \right] = (2^{-n})^k = 2^{-2m}.
\]

Hence,

\[
\Pr_{u_1, \ldots, u_k \in \{0,1\}^m} \left[ \exists r \in \{0,1\}^m, r \notin \bigcup_{i=1}^m (R + u_i) \right] \leq 2^m 2^{-2m} = 2^{-m} \ll 1.
\]

We now conclude the theorem. By the claims, we have that

\[
x \in L \iff \exists u_1, \ldots, u_k \in \{0,1\}^m \forall r \in \{0,1\}^m \bigvee_{i=1}^k M(x, r + u_i) = 1.
\]

7 Randomness as a proof tool

Randomness also gives us a lot of power to prove theorems. We will illustrate it by the example of good codes.

**Definition 7.1.** The hamming weight of \( x, y \in \{0,1\}^n \) is the number of coordinates on which they differ.

**Definition 7.2.** An \((n, k, d)\) binary code is a set \( C \subset \{0,1\}^n \) of size \( |C| = 2^k \) such that for any distinct \( x, y \in C \), their hamming weight is at least \( d \).

We think of a code as encoding \( k \) bits to \( n \) bits, while potentially allowing for error detection and correction.
Claim 7.3. An \((n,k,d)\) code allows to detect \(d-1\) errors, and to correct \((d-1)/2\) errors.

We say a code is good if \(k,d\) are linear in \(n\).

Lemma 7.4. There exist \((n,k = \alpha n, d = \beta n)\) codes for some \(\alpha, \beta > 0\) and any large enough \(n\).

Proof. Let \(C\) be a random subset of \(\{0,1\}^n\) of size \(2^k\). \(C\) is not a good code if it contains \(x, y\) of hamming weight at most \(d = \beta n\). The number of such pairs \(x, y\) is

\[
2^n \cdot \sum_{i=1}^{d} \binom{n}{i} \approx 2^{(1+H(\beta))n},
\]

where \(H(\beta) = \beta \log(1/\beta) + (1 - \beta) \log(1/(1 - \beta))\). The probability any fixed pair \(x, y\) is in \(C\) is

\[
\Pr_{C}[\{x, y\} \in C] = \frac{\binom{2^k}{2}}{\binom{2^n}{2}} \approx 2^{2(k-n)} = 2^{-2(1-\alpha)n}.
\]

Hence, the probability that \(C\) contains a "bad" pair \(x, y\) of hamming distance at most \(\beta n\) is

\[
2^{((1+H(\beta)) - 2(1-\alpha))n} = 2^{(H(\beta) + \alpha - n)}.
\]

For \(\beta \ll 1\) we have \(H(\beta) \approx \beta \log(1/\beta)\). Choosing \(\alpha, \beta > 0\) small enough so that \(H(\beta) + \alpha < 1\) gives that w.h.p \(C\) is a \((n, k = \alpha n, d = \beta n)\) code. \(\square\)