1 Randomized Algorithms

Before introducing derandomization, let us focus on first the concept of deterministic algorithm: given $X$ input bits and an Algorithm $A$, the output is either 0 or 1 (assume the output is binary). Hopefully, $A$ can be done in polynomial time, namely, $t \leq n^k$. For the randomized algorithm case, the algorithm is still deterministic, but the input contains some additional random bits. The two concepts are shown in Figure 1 and Figure 2 separately.

For example, Miller-Rabin Primality testing is a randomized algorithm. Before we go further, let’s introduce BPP.

**Definition 1.** BPP is the class of problems that are solvable in polynomial time with a randomized algorithm, namely, $\forall x$, $\Pr_r[ A(x, r) \text{ is correct}] \geq \frac{3}{4}$.

**Remark 1.** $\frac{3}{4}$ can be any $c \in (\frac{1}{2}, 1)$. To make it $1 - \epsilon$, we run $A(x, r)$ $O(\log(\frac{1}{\epsilon}))$ times independently, and take the majority vote.

**Remark 2.** If $A$ only takes $O(\log n)$ random bits, it’s trivial to make $A$ deterministic, simply try all $2^{O(\log n)} = \text{poly}(n)$ possible random bits, and take the majority vote.
2 Derandomization

The ultimate goal of derandomization is to address the question whether \( \text{BPP} = \text{P} \), i.e. how to make \( \mathcal{A} \) deterministic even if it use \( \omega(\log n) \) random bits? However, we are also interested in reducing the number of random bits used by a randomized algorithm.

2.1 Pseudo-random Generator (PRG)

**Definition 2.** Let \( \mathcal{C} \) be a class of functions. \( f : \{0, 1\}^n \rightarrow \{0, 1\}, G : \{0, 1\}^l \rightarrow \{0, 1\}^n (l < n) \) is an \( \epsilon \)-PRG for \( \mathcal{C} \) with seed length \( l \). If

\[
\forall f \in \mathcal{C} \quad \left| \Pr_{r \sim \{0,1\}^n} [f(r) = 1] - \Pr_{s \sim \{0,1\}^l} [f(G(s)) = 1] \right| < \epsilon
\]

Then, we say \( G \) \( \epsilon \)-fools \( \mathcal{C} \). Ideally, \( G(s) \) should be deterministically computable in polynomial time.

**Intuition:** Functions in \( \mathcal{C} \) can not distinguish between distribution \( \{G(s)\}_{s \sim \{0,1\}^l} \) and uniform distribution \( \{0, 1\}^n \). However, \( \{G(s)\} \) has a much smaller support.

**Example 1.** Say \( \mathcal{A} \) runs in \( n^{10} \) times, takes \( n \) random bits. Let \( \mathcal{C} \) be \( \{ f : \{0, 1\}^n \rightarrow \{0, 1\}, f \text{ is computable in } n^{10} \text{ time} \} \). If \( G \) is 0.1-fools \( \mathcal{C} \), then

\[
\Pr_{s \sim \{0,1\}^l} [\mathcal{A}(G(s)) \text{ correct}] \geq \Pr_{r \sim \{0,1\}^n} [\mathcal{A}(r) \text{ correct}] - 0.1 \geq \frac{3}{4} - 0.1 = 0.65
\]

\( \mathcal{A} \) is a deterministic algorithm. To enumerate \( s \) and take the majority vote runs in \( 2^l \text{poly}(n) \) time. If \( l = O(\log n) \), \( \mathcal{A} \) can be solved in \( \text{P} \).

**Theorem 3.** [Impagliazzo-Widgerson’97][1] Suppose \( \forall m \exists h_m : \{0, 1\}^n \rightarrow \{0, 1\} \) computable via a circuit of size \( 2^{100m} \), but not via any circuit of size \( 2^{0.001m} \), then \( \exists \) a PRG \( \epsilon \)-fools all poly-time algorithms with seed length \( O(\log m) \), i.e. \( \text{BPP} = \text{P} \) (The assumption is stronger than \( \text{P} \neq \text{NP} \), but believable).

**Intuition.** A function is hard to compute \( \Rightarrow \) looks random to Turing Machines with less time resource \( \Rightarrow \) fools the TMs.

2.2 \( k \)-wise Independent PRGs

**Definition 4.** \( G \{0,1\}^l \rightarrow \{0,1\}^n \) is \( k \)-wise independent, if \( \forall i \in [n] \Pr_s[(G(s))i = 1] = \frac{1}{2} \), and \( \forall 1 \leq i_1 < i_2 < \cdots < i_k \leq n \), the distribution \( \{(G(s))_{i_1}, (G(s))_{i_2}, \ldots (G(s))_{i_k}\}_s \) is uniform on \( \{0, 1\}^k \).
Construction of pairwise independent PRGs $G : \{0, 1\}^l \rightarrow \{0, 1\}^{2^l-1}$ is defined as $[G(s)]_v = (s, v) \mod 2 \forall v \in \{0, 1\}^l$, $v \neq \vec{0}$.

**Proof.** $\forall v \neq \vec{0} : \Pr_{s\in\{s, v\}}(s, v) \mod 2 = 1 = \frac{1}{2}$, and $\forall v \neq v_2 : \Pr_{s\in\{s, v_1\}}(s, v_2) \mod 2 \neq (s, v_2) \mod 2 \neq 0] = \frac{1}{2}$. $\square$

**Theorem 5.** [Alon-Babai-Itai’85][2] \( \forall k \leq n, \exists \) \( \text{poly}(n) \)-time computable \( k \)-wise independent generator with \( l = \lceil \frac{k}{2} \rceil \log n + O(1) \).

Let’s derandomize the following algorithm for MaxCut.

**MaxCut:** Given $G = (V, E)$, find $S \subseteq V$ to maximize $|\text{edges}(S, V - S)|$

**Algorithm:** For each $i \in V$, toss $r_i \in \{0, 1\}$, $i \in S$ iff. $r_i = 1$.

**Analysis:** $\mathbb{E}[\text{edges}(S, \bar{S})] = \mathbb{E}\sum_{(i,j) \in E} 1[r_i \neq r_j] = \sum_{(i,j) \in E} \Pr_{r}[r_i \neq r_j] = \sum_{(i,j) \in E} 1 = \frac{|E|}{2}$.

**Observation:** $r \in \{0, 1\}^n$ be pairwise independent suffice for the analysis. Use $r \in G(s)$ where $s \in r \in \{0, 1\}^{\log n}$ and $G$ is a pairwise independent PRG. Enumerating $s$ can be finished in polynomial time.

### 2.3 \( \epsilon \)-Bias Generators

**Definition 6.** $G : \mathbb{F}_2^l \rightarrow \mathbb{F}_2^n$ is \( \epsilon \)-biased PRG if $\forall w \in \mathbb{F}_2^n$, $w \neq 0$, $\Pr_{s \sim \mathbb{F}_2^l}[w \cdot G(s) = 1] \in [\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{2}]$.

**Theorem 7.** [NN’93][3] $l = O(\log \frac{n}{\epsilon})$ is achievable with poly-time computable.

**Theorem 8.** [AGHP’92][4] $l = 2\log \frac{n}{\epsilon} + O(1), O(\frac{n^2}{\epsilon^2})$-time computable.

**Application.** Input: $A, B, C \in \mathbb{F}_2^{n \times n}$. Goal: check $AB = C$ in $O(n^2 \cdot [\text{input size}])$ time.

**Algorithm:** choose $y \sim \mathbb{F}_2^n$ uniformly, check if $$(AB)y = Cy$$

Note that the calculation of $By$ needs $O(n^2)$. Similarly, calculation of $Cy$ and $A(By)$ need $O(n^2)$, too.

**Analysis:** $AB = C \Rightarrow \Pr[(AB)y = Cy] = 1$. On the contrary, $AB \neq C \Rightarrow D = AB - C$ has more than one non-zero rows. Let $D_i$ be $i$th row, $D_i \neq 0$, then

$$\Pr[(AB)y \neq Cy] = \Pr[Dy \neq \vec{0}] \geq \Pr[D_i^T \neq \vec{0}] = 1$$

Uses $O(n^2)$ time, $n$ is the length of random bits. If $y$ is the output of a 0.1-biased PRG, then

$$\Pr[D_i^T = 1] \geq \frac{1}{2} - \frac{0.1}{2} = 0.45 \Rightarrow O(n^2), O(\log n) \text{ random bits.}([AGHP’92])$$
Construction of $\epsilon$-biased generator

By definition, let $\text{Enc} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^l$ be $\forall w \in \mathbb{F}_2^n$, $s \in \mathbb{F}_2^l$, we have,

$$\text{Enc}(w)_s = G(s) \cdot w$$

Enc is the encoder of a linear code, with generator matrix like:

$$
\begin{bmatrix}
G(s_1)^T \\
G(s_2)^T \\
\vdots \\
G(s_n)^T
\end{bmatrix}
$$

By $\Pr_{s \sim \mathbb{F}_2} [G(s) \cdot w = 1] \in [\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{2}]$, $\forall w \neq 0$, all non-zero codewords has a relative weight $[\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{2}]$ $\Rightarrow$ relative distance $\geq \frac{1}{2} - \frac{\epsilon}{2}$. The rate is $\frac{n}{2^l}$.

2.4 Concatenation of Reed-Solomon and Hadamard codes

Let $C_1$ be Reed-Solomon code with parameter $[q, \epsilon, (1 - \epsilon)q]_q$, $C_2$ be Hadamard code with parameter $[q, \log_2 q, \frac{1}{2}q]_2$, and $C$ be the concatenation of $C_1$ and $C_2$, as shown in Figure 3.

![Figure 3: Concatenation of $C_1$ and $C_2$](image)

Claim 1. $C$ is a linear code for $q = 2^k$ ($k \in \mathbb{N}$) and appropriate encoder of $C_2$.

Claim 2. $C$ has a block-length of $q^2$, and the dimension of $C$ is $\epsilon q \log_2 q$.

Claim 3. $\forall$ non-zero codeword $c \in C$, $c$ has Hamming weight $[(1 - \epsilon)q \cdot \frac{1}{2}q, \frac{1}{2}q^2]$.

Proof. $\forall$ non-zero $u \in C_1$, $wt(u) \in [(1 - \frac{\epsilon}{2})q, q]$. For each non-zero $u_i$ encoded in $C_2$ has a weight of exactly $\frac{1}{2}q$. Therefore the weight of $c$ is $wt(u) \cdot \frac{1}{2}q \in [(1 - \epsilon)q \cdot \frac{1}{2}q, \frac{1}{2}q^2]$.

Finally, the $\epsilon$-generator has parameters:

$$2^l = q^2, \quad n = \epsilon q \log_2 q$$

$$\Rightarrow l = 2 \log_2 q, \quad n = \frac{\epsilon}{2} \cdot l \cdot 2^l < \frac{\epsilon}{2} \cdot 2^l$$

$$\Rightarrow l \leq 2 \log \frac{n}{\epsilon}.$$
Lecture 16: Class of Randomized Algorithms and Derandomization

References


