**6.842 Randomness and Computation** March 14, 2022

# Lecture 13

*Lecturer: Ronitt Rubinfeld* *Scribe: Guanghao Ye*

In this lecture, we will cover the following topics:

- Saving random bits via random walk
- Linearity Testing and Self-Correcting

# 1 Saving random bits via random walk

## 1.1 Linear Algebra

Recall from the last lecture, we have the following theorem about the random walk matrix:

**Theorem 1.** *If matrix $P \in \mathbb{R}^{n\times n}$ is a symmetric stochastic matrix, then there exists eigenvectors $v^{(1)}, \ldots, v^{(n)}$ which forms a orthonormal basis with corresponding eigenvalues $\lambda^{(1)}, \ldots, \lambda^{(n)}$ satisfying $1 = \lambda_1 \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$.*

As a result of theorem above, we have the following facts:

**Fact 2.** *The largest eigenvalue of $P$ is 1 with corresponding eigenvector $\frac{1}{\sqrt{n}}(1, 1, \ldots, 1)$.*

**Fact 3.** *For any vector $w \in \mathbb{R}^n$, we can find a decomposition of $w$ into the orthonormal basis formed by the eigenvectors $v^{(1)}, \ldots, v^{(n)}$ by the following formula:*

$$w = \sum_{i=1}^{n} \alpha_i v^{(i)} \qquad \text{where } \alpha_i = \langle w, v^{(i)} \rangle.$$

*Moreover,*

$$\|w\|_2 = \sqrt{\sum_{i=1}^{n} \alpha_i^2}.$$

**Fact 4.** *The $k$-th power of $P$ has same eigenvectors as $P$, $v^{(1)}, \ldots, v^{(n)}$, with eigenvalues $\lambda_1^k, \ldots, \lambda_n^k$.*

## 1.2 Saving randomness via Expander graph

For the simplicity, we consider the problems with one-sided error. Let $L$ be a language in RP, where there is an algorithm $\mathcal{A}$ deciding $L$ such that

1. For any $x \in L$, $\Pr[\mathcal{A}(x) = 1] \geq 0.99$
2. For any $x \notin L$, $\Pr[\mathcal{A}(x) = 0] = 1$.

Recall our previous lectures on reducing error for problems in RP, we have following result: Suppose $\mathcal{A}$ using $r$ many random bits, then for getting error on the order $2^{-k}$,

1. Naive methods: takes $O(kr)$ many random bits
2. Pairwise independence: takes $O(k + r)$ many random bits

In this lecture, we will show how to achieve the same result but only using $r + O(k)$ many random bits.

The idea is to find a connected graph $G$ with constant degree and self-loops, and $|\lambda_2| \leq 1/10$ with $2^r$ many nodes, so that we can treat each node of the graph as a $r$-bit string and use that as our randomness.

**Lemma 5.** *There exists a connected graph $G$ with self-loops on $2^r$ many nodes that satisfies:*

- *Each node has constant degree $d$.*
- *The random walk matrix on $G$ has $|\lambda_2| \leq 1/10$.*

**Remark** The first property of lemma above directly implies that the stationary distribution of random walk on graph $G$ is the uniform distribution.

Now, we state our algorithm:

**Algorithm 1** Error Reduction for RP

```
Input: Graph G, Algorithm A, input x
Pick a start node w ∈ {0,1}^r uniform at random
repeat
    Let w be a random neighbor of w
    Run A(x) using w as the random bits
    if A_w(x) outputs x ∈ L then
        Output x ∈ L
    end if
until k times
Output: x ∉ L
```

The algorithm above uses $r$ random bits to pick the initial $w$, and for each iteration, it need $\log d$ bits to choose the neighbor. Thus, the total amount of randomness used in the algorithm above is $r + O(k)$ since $d$ is constant.

**Claim 6.** *The error probability of the algorithm above is at most $5^{-k}$ for $x \in L$ and 0 for $x \notin L$.*

*Proof.* For note that for $x \notin L$, $\mathcal{A}$ will never accepts, thus the algorithm above will always output $x \notin L$.

For $x \in L$, let $B \stackrel{\text{def}}{=} \{w \mid \mathcal{A}(x)$ with random bits $w$ says "$x \notin L$"$\}$. Since $\mathcal{A}$ corrects with probability at 0.99, we have $|B| \leq 2^r/100$.

To use the spectral property of the random walk matrix, we define the diagonal matrix $N_w$ such that

$$N_{w,w} = \begin{cases} 1 & \text{if } w \in B \\ 0 & o.w. \end{cases}$$

Let $q$ is a probability distribution, $qN$ deletes weight on good nodes. That is picking $w$ according to $q$, we have $\|qN\|_1 = \Pr_{w \in q}[w \text{ is bad}]$.

The diagonal matrix $N$ can be compose with the transition matrix $P$, we note that

$$\|qPN\|_1 = \Pr_{w \in q}[\text{start at } w, \text{ take a step, and land on a bad node.}]$$

$$\|qPNPN\|_1 = \Pr_{w \in q}[\text{start at } w, \text{ take two step, and land on bad nodes twice.}]$$

Generally, we have

$$\|q(PN)^i\|_1 = \Pr_{w \in q}[\text{start at } w, \text{ take } i \text{ step, and land on bad nodes all the time.}]$$

**Lemma 7.** *For any vector $\pi$, we have*

$$\|\pi PN\|_2 \leq \frac{1}{5}\|\pi\|_2.$$

Before we prove the lemma, we see how this implies the theorem. Let $\pi$ be the uniform distribution, we have

$$\begin{aligned}\Pr[\text{incorrect}] &\le \|\pi(PN)^k\|_1 \\ &\le \sqrt{2^r}\|\pi(PN)^k\|_2 \\ &\le \sqrt{2^r}\frac{1}{5^k}\|\pi\|_2 \\ &= \frac{1}{5^k}\end{aligned}$$

where the first step follows by Cauchy-Schwarz, the second step follows by applying the lemma above $k$ times, and the last step follows by $\pi$ is the uniform distribution. ■

Now, we prove the lemma we used above.

*Proof of Lemma 7.* Let $v_1, \ldots, v_{2^r}$ be eigenvectors of $P$, where $v_1 = \frac{1}{\sqrt{2^r}}(1, \ldots, 1)^\top$. Then, we can write

$$\pi = \sum_{i=1}^{2^r} \alpha_i v_i \tag{1}$$

Then, we have

$$\|\pi PN\|_2 = \left\|\sum_{i=1}^{2^r} \alpha_i v_i PN\right\|_2 = \left\|\sum_{i=1}^{2^r} \alpha_i \lambda_i v_i N\right\|_2 \le \underbrace{\|\alpha_1 \lambda_1 v_1 N\|_2}_{A} + \underbrace{\left\|\sum_{i=2}^{2^r} \alpha_i \lambda_i v_i N\right\|_2}_{B}$$

where the first step follows by (1), the second step follows by $v_i$ is eigenvectors, and the last step follows by triangle inequality.

Now, we bound $A$ and $B$ separately.

$$\begin{aligned}A &\stackrel{\text{def}}{=} \|\alpha_1 \lambda_1 v_1 N\|_2 \\ &= \|\alpha_1 v_1 N\|_2 \qquad (\text{since } \lambda_1{=}1) \\ &= |\alpha_1|\frac{1}{\sqrt{2^r}} \cdot \|\mathbf{1} \cdot N\| \qquad (\text{since } \|v\| = \frac{1}{\sqrt{2^r}}{=}1) \\ &= \frac{|\alpha_1|}{\sqrt{2^r}}\sqrt{\sum_{i \in B} 1} \qquad (\text{definition of } N) \\ &\le \frac{|\alpha_1|}{\sqrt{2^r}} \cdot \sqrt{\frac{2^r}{100}} \qquad (\text{since } |B| \le 2^r/100) \\ &= \frac{|\alpha_1|}{10} \\ &\le \|\pi\|/10 \qquad (\text{since } \|\pi\|_2 = \sqrt{\sum_i \alpha_i^2} \ge |\alpha_1|).\end{aligned}$$

To bound $B$:

$$\begin{aligned}
B &\stackrel{\text{def}}{=} \left\| \sum_{i=2}^{2^r} \alpha_i \lambda_i v_i N \right\|_2 \\
&\leq \left\| \sum_{i=2}^{2^r} \alpha_i \lambda_i v_i \right\|_2 \\
&= \sqrt{\sum_{i=2}^{2^r} (\alpha_i \lambda_i)^2} \\
&\leq \frac{1}{10} \sqrt{\sum_{i=2}^{2^r} (\alpha_i)^2} \qquad \text{(since } |\lambda_i| \leq |\lambda_2| < 1/10\text{)} \\
&\leq \frac{1}{10} \|\pi\|_2.
\end{aligned}$$

Hence, we have

$$\|\pi P N\|_2 \leq A + B \leq \frac{1}{5} \|\pi\|_2.$$

■

# 2 Linearity Testing

Now, we discuss the second of topic of lecture today: linearity testing.

**Motiviation.** How to check a program that is outputting the right answer? Instead of proving a program is correct, we want the program itself shows to us that it's outputting the correct answer. In the scheme of interactive proof, we know one can the graph isomorphism, and the question now is how about other classes of problems? For example, linear problems.

Given a program that suppose to compute some linear function, e.g., matrix multiplication, integer modulo, etc. We are given the oracle access to this program.

The claim is the program $f$ is computing some linear function, and we want to test if $f$ is linear fast.

Now, we formally define the class of linear functions:

**Definition 8.** *A function $f : G \to H$ is* linear *(homomorphism) if for any $x, y \in G$, we have*

$$f(x) +_H f(y) = f(x +_G y).$$

*Moreover, we say $f$ is* $\epsilon$-linear *if there exits a linear function $g$ such that $f$ and $g$ agree on at least $1 - \epsilon$ fraction of inputs in $G$, i.e.,*

$$\Pr_{x \in G}[f(x) = g(x)] \geq 1 - \epsilon.$$

The propose test is the following:

**Algorithm 2** Proposed linearity test

1: **repeat**
2: Pick random $x, y \in G$
3: Test if $f(x) +_H f(y) = f(x +_G y)$.
4: **until** ??? times

Now, the question is how many times to need to repeat.