6.5420 Randomness and Computation March 3, 2026

Lecture 7

Lecturer: Ronitt Rubinfeld *Scribe: Nikita Romanov*

This lecture covers:

1. Saving random bits in amplification via random walks
2. A randomized algorithm for approximate (two-multiplicative) Max-Cut

1 Saving random bits via random walk

1.1 Linear algebra warm-up

Recall the following theorem about the random walk matrix.

Theorem 1. *If $P \in \mathbb{R}^{n\times n}$ is a symmetric row-stochastic matrix, then there exist eigenvectors $v^{(1)}, \ldots, v^{(n)}$ forming an orthonormal basis with corresponding eigenvalues $\lambda^{(1)}, \ldots, \lambda^{(n)}$ satisfying*

$$1 = \lambda^{(1)} \geq |\lambda^{(2)}| \geq \cdots \geq |\lambda^{(n)}|.$$

In the same setting, we also have the following facts

Fact 2. *P has an eigenvector:*

$$\sqrt{\frac{1}{n}}(1, 1, \ldots, 1).$$

with the corresponding eigenvalue 1. This follows directly from symmetry and row-stochasticity, i.e. P need not correspond to a d-regular graph.

Fact 3. *For any vector $w \in \mathbb{R}^n$, we can decompose w in the orthonormal eigenbasis:*

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

Moreover,

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

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

1.2 Saving randomness via expander graphs

Our goal in this lecture is to save the number of random bits used in amplifying the success probability of some algorithm. For simplicity, we focus on the one sided-error case. This approach, however, readily generalizes to two-sided errors.

Let $L \in$ RP, meaning there exists a randomized algorithm $\mathcal{A}$ deciding L that uses $r(n)$ bits of randomness and satisfies:

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

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

We know that a simple way to reduce the failure probability of $\mathcal{A}$ below 2^{-k} is to run the algorithm k times and output "$x \in L$" if ever see $\mathcal{A}(x) = 1$ (with our algorithm $\mathcal{A}$, we'd actually reduce the error probability below 100^{-k}). This naive repetition draws independent r-bit strings and thus requires a total of $O(kr)$ random bits.

As we show below, one can achieve a similar success amplification using only

$$r + O(k)$$

random bits by performing a random walk on an expander graph G. We pick this graph to be d-regular and have 2^r vertices corresponding to different r-bit strings used by $\mathcal{A}$. The key idea that enables this saving of random bits is that if G satisfies certain properties, it suffices to make random decisions locally (i.e. among d neighbors) rather than globally (among 2^r options). In other words, the saving comes from the fact that we can choose r-bit strings in a dependent way.

While we don't prove it here, the good news is that such a graph G exists:

Lemma 5. *There exists an undirected connected graph G with 2^r vertices such that:*

- *Each vertex has constant degree d.*
- *The transition matrix P of G satisfies $|\lambda_2| \leq 1/10$.*
- *The transition matrix P is symmetric (follows from undirected + d-regular).*

Remark. The constant-degree and connectivity (irreducibility) imply that the stationary distribution of the random walk on G is unique and uniform.

Remark. $|\lambda_2| \leq 1/10$ implies that the connected undirected graph G is aperiodic.

So essentially, we trade off a small amount of space (to maintain our position in the walk on G) to reduce the number of fresh random bits needed for success amplification. Notably, we only need an explicit neighbor-computation procedure, avoiding the $O(2^r)$-size representation of G.

1.3 Algorithm and guarantees

Assuming G from Lemma 5, we now present the algorithm:

Algorithm 1 Success amplification for RP via random walk

```
Input: Graph G, algorithm A, instance x
Pick a start node w ∈ {0,1}^r uniformly at random
for t = 1 to k do
    w ← uniformly random neighbor of w
    Run A(x) using w as the random-bit string
    if A(x) accepts then
        Output x ∈ L
    end if
end for
Output x ∉ L
```

This uses r random bits to choose the initial w, and then $\log d$ bits per step to choose a neighbor. Since d is constant, total randomness is $r + O(k)$.

Claim 6. *For $x \in L$, the error probability is at most 5^{-k}, and for $x \notin L$ it is 0.*

Proof. For $x \notin L$, regardless of the vertices w visited during the random walk, the algorithm would output "$x \notin L$".

For $x \in L$, let $\mathcal{A}_w$ denote running algorithm $\mathcal{A}$ using w as the random input string. Also, define:

$$B \stackrel{\text{def}}{=} \{w \mid \mathcal{A}_w(x) \text{ incorrectly outputs " } x \notin L\text{"}\}$$

Since vertices of G correspond to 2^r possible input random strings and $\mathcal{A}(x)$ succeeds with probability at least 0.99, $|B| \leq \frac{2^r}{100}$.

To keep track of the probability that our random walk would incorrectly output "$x \notin L$", we also define a diagonal matrix $N \in \mathbb{R}^{2^r \times 2^r}$ indexed by vertices of G:

$$N_{w,w} \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } w \in B \text{ (i.e. "bad" string)}, \\ 0, & \text{otherwise.} \end{cases}$$

So given a distribution q over vertices of G, we have $\|qN\|_1 = \Pr_{w \sim q}[w \text{ is " bad"}]$. Let P denote the transition matrix of G. We note that :

$$\|q(PN)\|_1 = \Pr_{w \sim q}[\text{the vertex after one step is bad}]$$

Generally, after doing i steps on the graph G, we get:

$$\|q(PN)^i\|_1 = \Pr_{w \sim q}[\text{all } i \text{ visited vertices are bad}]$$

With the notation in hand, the key technical input is the following lemma, the proof of which we defer for later.

Lemma 7. *For any $\Pi \in \mathbb{R}^{2^r}$:*

$$\|\Pi PN\|_2 \leq \frac{\|\Pi\|_2}{5}$$

Let U_{2^r} denote the uniform distribution on 2^r vertices. Thus,

$$\begin{aligned} \Pr[\text{our algorithm produces incorrect output}] &= \|U_{2^r}(PN)^k\|_1 \leq \\ &\leq \sqrt{2^r}\|U_{2^r}(PN)^k\|_2 \leq \\ &\leq \frac{\sqrt{2^r}\|U_{2^r}\|_2}{5^k} \leq \\ &\leq \frac{1}{5^k} \end{aligned}$$

Where the first step follows from $\|\cdot\|_2 \leq \|\cdot\|_1 \leq \sqrt{\dim}\,\|\cdot\|_2$, second step from Lemma 7 (applied k times) and the last step from the fact that the original distribution U_{2^r} is uniform over 2^r nodes. This concludes the proof of the claim. □

Proof of Lemma 7. Since the transition matrix P of G is symmetric, it has an orthonormal eigenbasis $\{v_1, \ldots, v_{2^r}\}$ with $v_1 = \frac{1}{\sqrt{2^r}}(1, \ldots, 1)$ (note $\|v_1\|_2 = 1$) and $\lambda_1 = 1$. We are also given (Lemma 5) that $|\lambda_2| \leq 1/10$. Let us expand Π in that basis:

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

Therefore, expanding Π and using the triangle inequality:

$$\|\Pi PN\|_2 = \|\sum_{i=1}^{2^r} \alpha_i v_i PN\|_2 \leq \underbrace{\|\alpha_1 v_1 PN\|_2}_{A} + \underbrace{\|\sum_{i=2}^{2^r} \alpha_i v_i PN\|_2}_{B}$$

We upper bound A and B separately. For A, we use $v_1 P = v_1$ and the fact that v_1 is uniform:

$$\|\alpha_1 v_1 PN\|_2 = |\alpha_1| \cdot \|v_1 N\|_2 = \frac{|\alpha_1|}{\sqrt{2^r}} \sqrt{|B|} \leq \frac{|\alpha_1|}{\sqrt{2^r}} \sqrt{\frac{2^r}{100}} \leq \frac{\|\Pi\|_2}{10}$$

where the last inequality follows from $\|\Pi\|_2 = \sqrt{\sum_i |\alpha_i|^2}$.

Now, for B, using the fact that $\{v_i\}$ is an eigenbasis of P with $1 = \lambda_1 \geq |\lambda_2| \geq \cdots \geq |\lambda_{2^r}|$, we get:

$$\|\sum_{i=2}^{2^r} \alpha_i v_i PN\|_2 = \|\sum_{i=2}^{2^r} \alpha_i \lambda_i v_i N\|_2 \leq \|\sum_{i=2}^{2^r} \alpha_i \lambda_i v_i\|_2 \leq \sqrt{\sum_{i=2}^{2^r} |\alpha_i \lambda_i|^2} \leq |\lambda_2| \sqrt{\sum_{i=2}^{2^r} |\alpha_i|^2} \leq \frac{\|\Pi\|_2}{10}$$

where the second step follows from the contractility of ℓ^2 norm under multiplication by N and third from the orthonormality of $\{v_i\}$

Now, combining A and B together, we conclude the proof:

$$\|\Pi PN\|_2 \leq \underbrace{\|\alpha_1 v_1 PN\|_2}_{A} + \underbrace{\|\sum_{i=2}^{2^r} \alpha_i v_i PN\|_2}_{B} \leq \frac{\|\Pi\|_2}{10} + \frac{\|\Pi\|_2}{10} \leq \frac{\|\Pi\|_2}{5}$$

□

2 Approximate Max-Cut Algorithm

To set the stage for the next lecture, we briefly cover a simple approximate algorithm for finding the maximum cut of a graph.

Problem 8 (Finding Max-Cut of a graph)**.** *Given a graph $G = (V, E)$, the goal is to find a partition S, T of V that maximizes:*

$$\big|\{(u, v) \in E : u \in S, v \in T\}\big|$$

While the exact Max-Cut problem is NP-hard, one can find a good approximation by running the following algorithm.

2.1 Randomized approximate algorithm

Algorithm.

- Flip $n = |V|$ coins with outcomes $(r_1, \ldots, r_n)$ corresponding to either S or T.
- For each i, put vertex $v_i \in V$ on side r_i.

Analysis. For every edge $\{u, v\} \in E$, define the indicator random variable

$$\mathbf{1}_{u,v} \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } r_u \neq r_v \quad \text{(i.e., } u \text{ and } v \text{ are on opposite sides of the cut)}, \\ 0, & \text{otherwise.} \end{cases}$$

Then the cut size is:

$$\text{cut\_size} \;=\; \sum_{\{u,v\} \in E} \mathbf{1}_{u,v}.$$

By linearity of expectation,

$$\begin{aligned}\mathbb{E}[\text{cut\_size}] &= \mathbb{E}\left[\sum_{\{u,v\}\in E} \mathbf{1}_{u,v}\right] = \sum_{\{u,v\}\in E} \mathbb{E}[\mathbf{1}_{u,v}] = \sum_{\{u,v\}\in E} \Pr[\mathbf{1}_{u,v} = 1] \\ &= \sum_{\{u,v\}\in E} \Pr\Big[(r_u = 1 \wedge r_v = 0) \ \vee \ (r_u = 0 \wedge r_v = 1)\Big] \\ &= \sum_{\{u,v\}\in E} \left(\frac{1}{4} + \frac{1}{4}\right) = \frac{|E|}{2}.\end{aligned}$$

Thus in expectation, half of the edges cross the cut. This implies that there exists at least one cut of size $\geq |E|/2$. This is great since the cut size can be at most $|E|$, so the randomized algorithm gives a multiplicative approximation (in expectation) to within a factor of two.

2.2 Derandomization of Max-Cut via enumeration

Recall that to derandomize an algorithm $\mathcal{A}$ via enumeration, one runs $\mathcal{A}$ with every possible random string of length $r(n)$ and output the majority answer. Random Max-Cut has $r(n) = n$ and thus would require enumerating 2^n options, which quickly becomes intractable.

Question Can we still derandomize Max-Cut via enumeration by somehow reducing $r(n)$?

As we'll see in the next lecture, the answer is yes. The idea is to find a subset of all random strings $S = \{0,1\}^{r(n)}$ that "works" and then enumerate over them.