Lecture 7

more on random walks
maybe begin pairwise independence motivation

- saving random bits in amplification via random walks

- if time: begin pairwise independence
  start with algorithm for max-cut that we will derandomize next time via pairwise independence

Reducing Randomness via Random Walks:

For language L,

let A be algorithm using r bits of randomness s.t.

1-sided error

(1) $\forall x \in L \quad \Pr_{w \in A\text{'s coins}}[A_w(x)=1] \geq 99/100$ usually correct

(2) $\forall x \notin L \quad \Pr_{w \in A\text{'s coins}}[A_w(x)=0] = 1$ always correct

To get error $< 2^{-k}$

Method	# random bits used
1) run k times on independently chosen bits & output "$x \notin L$" if see 0 else output "$x \in L$".	$k \cdot r$
2) today: use random walks to choose bits (↑ not independent)	$r + O(k)$

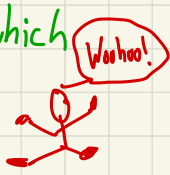
Plan

- $\forall$ (random) string $w \in \{0,1\}^r$, assign it to node in graph G ← we pick!

- picking random n-bit string
  $\iff$ picking random node in G
  easier?

picking several random n-bit strings
$\iff$ picking several random nodes in G
easier?

picking several strings, one of which is "good"
$\iff$ picking several nodes, one of which is "good"
Easier!!



The graph G: ← we get to pick G!!!

- constant degree d-regular, connected, nonbipartite
- transition matrix P for r.w. on G has $|\lambda_2| \leq \frac{1}{10}$

since <1 implies G is connected! → λ_2 + d-reg ⇒ stat dist π is uniform

- # nodes $= 2^r$ corresponds to all possible choices of r random bits

The Algorithm

random bits

- Pick random start node $w \in \{0,1\}^r$ — r
- Repeat k times:

  $w \leftarrow$ random nbr of w

  run $A_w(x)$ with w as random bits.

  If $A_w(x) = 0$, output "$x \notin L$" & halt

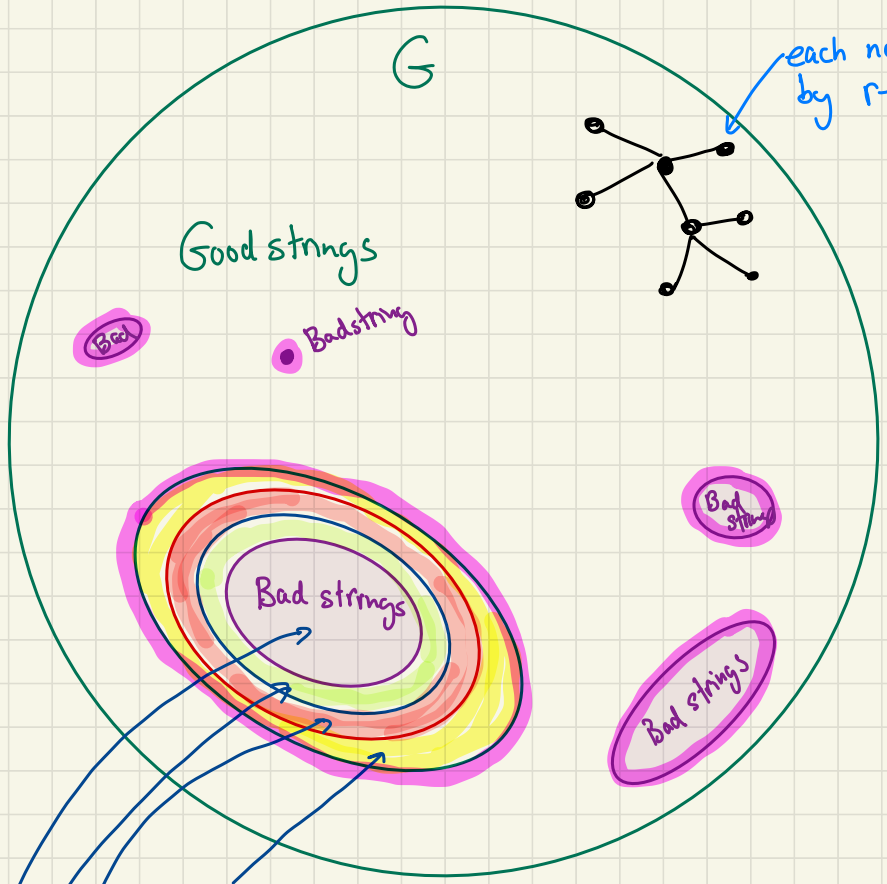
  else continue

  $O(1) \times k$ (↑ d is const, ↑ # loops)
- Output "$x \in L$"

total: $r + O(k)$

Behavior: Claim: error of new algorithm is $\leq (\frac{1}{5})^k$ for $x \in L$

(still 0 error for $x \notin L$)



Main Idea

unlikely to pick start location that is likely bad after k-steps

bad case: walk only on "bad strings" & never reach good strings

why is this possible if G arbitrary? e.g. line

↑ λ_2 is close to 1

Proof of Claim

$x \notin L$: algorithm never errs (no bad strings)

$x \in L$:

most random bits say $x \in L$ ($A(x)=1$): $\geq \frac{99}{100} \cdot 2^r$

define $B \equiv \{ w \mid A_w(x)$ with random bits w is incorrect. i.e. says $x \notin L \}$

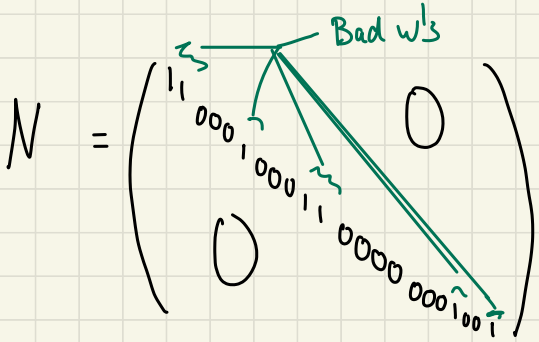
"bad w's"

$$|B| \leq \frac{2^r}{100}$$

need lin. alg. way of describing walks that stay in bad set:

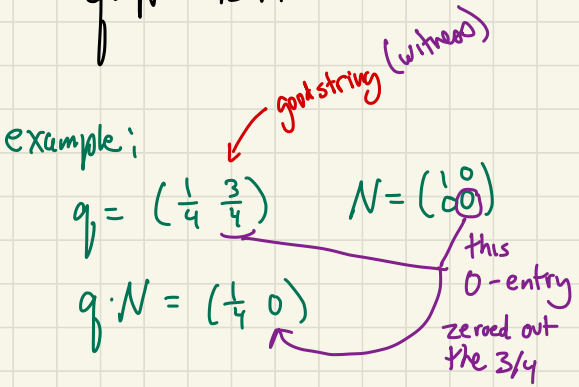
define N diagonal matrix

$$N_w = \begin{cases} 1 & \text{if } w \in B \leftarrow \text{incorrect} \\ 0 & \text{o.w.} \leftarrow \text{correct} \end{cases}$$



For q <u>any</u> probability dist:

$q \cdot N$ is??



$q \cdot N$ deletes weight that finds a witness to $x \notin L$

$$\|q \cdot N\|_1 = \Pr_{w \in q}[w \text{ is bad}]$$

Can compose:

$$\|q \cdot PN\|_1 = \Pr_{w \leftarrow q}[\text{start at } q, \text{ take a step \& land on "bad"}]$$

$$\vdots$$

$$\|q(PN)^i\|_1 = \Pr_{w \leftarrow q}[\text{start at } q, \text{ take } i \text{ steps \& each is "bad"}]$$

ignores whether start node bad. this just hurts us, so ok to ignore.

Lemma $\forall \pi \quad \|\pi PN\|_2 \leq \frac{1}{5}\|\pi\|_2$

First: how do we use lemma?

answer incorrect only if always see bad w/ $\frac{1}{2}$

$$\Rightarrow \Pr[\text{incorrect}] \leq \|p_0 (PN)^k\|_1$$

$$\leq \sqrt{2^r}\, \|p_0 (PN)^k\|_2$$

since $\|p\|_1 \leq \sqrt{\text{domain size}} \cdot \|p\|_2$

$$\leq \sqrt{2^r}\, \|p_0\|_2 \left(\frac{1}{5}\right)^k$$

apply lemma k times

$$= \frac{1}{\sqrt{2^r}}$$

since start at uniform & L_2 norm of uniform $= \sqrt{\sum \left(\frac{1}{2^r}\right)^2} = \sqrt{\frac{1}{2^r}}$

$$= \left(\frac{1}{5}\right)^k$$

✓ great! our algorithm works!

Proof of lemma:

let $V_1 .. V_{2^r}$ be e-vecs of P that are orthonormal & V_1 is s.t. $\|V_1\|_2 = 1$ (so $V_1 = \left(\frac{1}{\sqrt{2^r}}, \dots, \frac{1}{\sqrt{2^r}}\right)$)

Then $\pi = \sum_{i=1}^{2^r} \alpha_i V_i$

note: 1) $\|\pi\|_2 = \sqrt{\sum \alpha_i^2}$ by (*) proved previously

2) $\forall\, w \quad \|wN\|_2 = \sqrt{\sum_{i \in B} w_i^2} \leq \sqrt{\sum_i w_i^2} = \|w\|_2$

So:

$$\|\Pi PN\|_2 = \left\| \sum_{i=1}^{2^n} \alpha_i v_i PN \right\|_2$$

since any Π is lin comb of basis vectors

$$= \left\| \sum_{i=1}^{2^n} \alpha_i \lambda_i v_i N \right\|_2$$

$$\leq \underbrace{\| \alpha_1 \lambda_1 v_1 N \|_2}_{(A)} + \underbrace{\left\| \sum_{i=2}^{2^n} \alpha_i \lambda_i v_i N \right\|_2}_{(B)}$$

Cauchy-Schwarz

bound (A):

$$\| \alpha_1 \lambda_1 v_1 N \|_2 = \| \alpha_1 v_1 N \|_2$$

since $\lambda_1 = 1$

$$= |\alpha_1| \cdot \sqrt{\sum_{i \in B} \left(\frac{1}{\sqrt{2^r}}\right)^2}$$

since $v_1 = \left(\frac{1}{\sqrt{2^r}}, \dots, \frac{1}{\sqrt{2^r}}\right)$ & $N = \begin{pmatrix} \square & & 0 \\ & \ddots & \\ 0 & & \square \end{pmatrix}$

uses that uniform dist is unlikely to be on a bad string

$$= |\alpha_1| \cdot \sqrt{\frac{|B|}{2^r}}$$

$$\leq \frac{|\alpha_1|}{10}$$

since $\frac{|B|}{2^r} \leq \frac{1}{100}$

$$\leq \frac{\|\Pi\|_2}{10}$$

since $\|\Pi\|_2 = \sqrt{\sum_{i=1} \alpha_i^2}$

bound (B):

$$\left\| \sum_{i=2}^{2^r} \alpha_i \lambda_i v_i N \right\|_2 \le \left\| \sum_{i=2}^{2^r} \alpha_i \lambda_i v_i \right\|_2 \quad \text{from note}$$

$$= \sqrt{\sum (\alpha_i \lambda_i)^2}$$

$$\le \sqrt{\sum \alpha_i^2 \cdot \left(\frac{1}{10}\right)^2} \quad \lambda_i \le 1/10$$

$$\le \frac{1}{10} \cdot \|\pi\|_2$$

uses "mixing" of v_i's for $i > 2$. These could be "heavy" in bad areas, but won't stay for long!!

So: $\|\pi P N\|_2 \le \frac{\|\pi\|_2}{5}$ ∎

Pairwise Independence & Derandomization

Let's start with a simple algorithm for MaxCut:

Max Cut:

given: $G = (V, E)$

output: partition V into S, T

to maximize $|\{(u,v) \mid u \in S, v \in T\}|$ } NP hard

size of S-T cut

Randomized Algorithm:

Flip coins $r_1 \cdots r_n$

Put node i on side r_i to get S, T

← if $r_i = 0$ add i to S
else $r_i = 1$ add i to T

Analysis:

u, v on opposite sides of cut

Let $1_{u,v} = \begin{cases} 1 & \text{if } r_u \neq r_v \\ 0 & \text{o.w.} \end{cases}$

$$\text{Cut size} = \sum_{(u,v) \in E} 1_{u,v}$$

$$E[\text{cut size}] = E\left[\sum_{(u,v) \in E} 1_{u,v}\right]$$

$$= \sum_{(u,v) \in E} E[1_{u,v}] = \sum_{(u,v) \in E} Pr[1_{u,v} = 1]$$

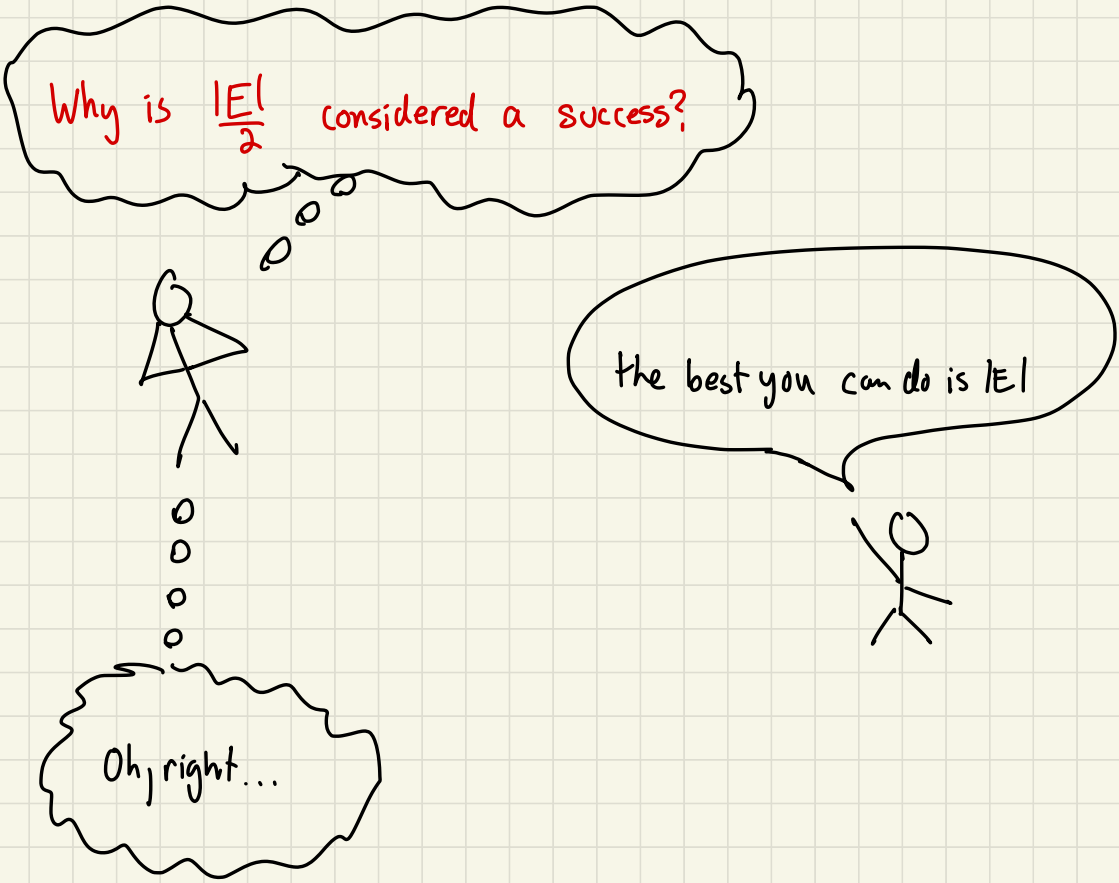
$$= \sum_{(u,v) \in E} Pr\left[(r_u = 1 \text{ and } r_v = 0) \text{ or } (r_u = 0 \text{ and } r_v = 1)\right]$$

$$= |E| \cdot \left[\frac{1}{4} + \frac{1}{4}\right] = \frac{|E|}{2}$$

So expect $\frac{1}{2}$ the edges to cross cut!

Note: $E[\text{cut size}] = \frac{|E|}{2} \Rightarrow \exists$ cut of size $\frac{|E|}{2}$

- $E[\text{cut size}]$: average cut size produced by algorithm
- cut of size $\frac{|E|}{2}$: must be one that is at least as big as the average



Gives multiplicative approximation to within a factor of 2

Previous lecture

Derandomization via Enumeration

Given: probabilistic algorithm A + input x

Algorithm:

Run A on every possible random string of length $r(n)$

Output majority answer

Randomized Max Cut Algorithm:

Flip coins $r_1 \cdots r_n$

Put node i on side r_i to get S, T

Derandomization: first attempt

use "derandomization via enumeration"

Run A on every possible random string of length $r(n)$

Output majority answer

here $r(n) = n$, so need 2^n runs of A 

Hope: reduce $r(n)$?

still use "derandomization via enumeration"

find subset $S \subseteq \{0,1\}^{r(n)}$ of random strings that "works"

$\Rightarrow$ only enumerate over S