6.842 Lecture 3

- The Lovasz Local Lemma (recap + finish)
- Polynomial Identity Testing
Lovasz Local Lemma: Recap & finish

**Goal:** Show that possibly no bad events happen!

**Possible tools:**
- if independent, then obvious
- if not independent, use union bound
- What if $A_i$'s have "some" independence?

**def.** A "independent" of $B_1, B_2, ..., B_k$ if

$$\forall J \subseteq [k] \text{ then } \Pr\left[A \land \bigwedge_{j \in J} B_j \right] = \Pr\left[A\right] \cdot \Pr\left[\bigwedge_{j \in J} B_j \right]$$

**Note:** $[k]$ means $\{1, ..., k\}$

**def.** $A_1, ..., A_n$ events

$D = (V, E)$ with $V = [n]$ is "dependency digraph of $A_1, ..., A_n$"

if each $A_i$ independent of all $A_j$ that are not neighbors in $D$ (i.e. all $A_j$ st. $(ij) \in E$)
**Lovász Local Lemma**  (symmetric version)

$A_1,\ldots, A_n$ events s.t. $\Pr(A_i) \leq p$ $\forall i$

with dependency digraph $D$ s.t. $D$ has max degree $\leq d$.

If $e_p(d+1) \leq 1$, then

$$\Pr\left[ \bigwedge_{i=1}^{n} \overline{A_i} \right] > 0$$

**Application**

**Thm.** Given $S_1,\ldots, S_m \subseteq X$ $|S_i| = l$

each $S_i$ intersects at most $d$ other $S_j$'s

If $e_p(d+1) \leq 2^{l-1}$

then can $2$-color $\overline{X}$ such that

each $S_i$ not monochromatic

ie. $H$ is hypergraph with $m$ edges, each containing $l$ nodes & each intersecting $\leq d$ other edges
Stronger assumptions:

(1) For today, assume \( l, d \) constants.

(2) Binary Entropy: \( H(x) = -x \log_2 x - (1-x) \log_2 (1-x) \)

Let \( p = 2 \cdot 2^{(H(x)-1) \cdot l} \)

\[ e \cdot \frac{1}{d+1} < \frac{1}{2} \]

(3) \( 2 e (d+1) < 2^\alpha n \)
Algorithm: Given $S_1, \ldots, S_m \subseteq X$, $|S_i| = l \ \forall i$

First pass:

for each $j \in X$ pick color red/blue via coin toss

$S_i$ is "bad" if $\leq \alpha \cdot l$ reds or $\leq \alpha \cdot l$ blues

$B = \{ S_i \mid S_i$ is bad $\}$

1st pass is successful if all "connected components" of $B$ are $\leq d \log m$

(if not successful, retry)

Second Pass:

Brute force each connected component (w/o violating their nbrs)
After 1st pass: orange Si's are "good", red Si's are "bad"

Some questions:

1. Why is output legal? What if changing Si's in B makes Si & B monochromatic?
2. How fast is pass 2?
3. How many time to repeat pass 1?

No way this is fast!
Why is output legal?

First pass:
For each $j \in X$ pick color red/blue via coin toss

$S_i$ is "bad" if $\leq \alpha_\ell$ reds
or $\leq \alpha_\ell$ blues

$B = \{ S_i \mid S_i$ is bad $\}$

Pass successful if all "connected components"
of bad $S_i$'s are $\leq d \log m$
(if not successful, retry)

Second Pass: Brute force each connected component

If $S_i$ not bad + < $\alpha n$ nodes in bad nbrs
then $S_i$ will still be bichromatic after recoloring.

If $S_i$ not bad + has $\geq \alpha n$ nodes in bad nbrs,
then $\geq \alpha n$ nodes get recolored

Main idea:
remaining subproblems each have property
that all remaining sets have enough uncolored points
so that LLL
$\Rightarrow$ soln exists

If $S_i$ not bad + < $\alpha n$ nodes in bad nbrs
then $S_i$ will still be bichromatic after recoloring.

If $S_i$ not bad + has $\geq \alpha n$ nodes in bad nbrs,
then $\geq \alpha n$ nodes get recolored

- if recolored randomly, $\Pr[S_i \text{ is monochrom}] < 2^{-\alpha n}$
- using LLL
  - assume $2e(d+1) < 2^{\alpha n}$

This was assumption 3

$\Rightarrow$ solution exists!
Main idea:
Components small $\Rightarrow$ involve few sets
$\Rightarrow$ involve few elements (since assume $l$ is $O(1)$)
$\Rightarrow$ can brute force each one separately

How fast is Pass 2?

size of Surviving components $\leq O(d \log m)$

$\# \text{ settings to vars in a surviving component} \leq 2^l O(d \log m)$

$= O(l^d)$

$= m$

total time: $\# \text{ surviving components} \times m = m$

if $dl$ constant: poly($m$) time $\ast$ assumption

else, recurse on components
How many times to repeat pass 1?

Complications:
- Need "refined" def of "connected component" for pass 2 to work
  Why? since need to recolor some non-bad sets that neighbor bad sets
Let's be more careful in our defn. of conn components.

Hypergraph: nodes for each \( x \in X \)

hyperedge \( S_i \) corresponds to subset of \( X \)

( all \( |S_i| = 2 \Rightarrow \) usual notion of graph )

Hyperedge \( S_i \):

Dependency digraph: nodes for each \( S_i \)

edge between \( S_i \) & \( S_j \) if intersect

Not directed in this case

Dependency digraph:

All \( S_i \)s:

Piece of Dependency Digraph:

assumption \( \Rightarrow \) this graph has degree \( \leq d \)
After 1st pass: orange $S_i$'s are "good", red $S_i$'s are "bad"

How should we define "connected component"?

Try 1: use dependency graph
degree of nodes by assumption
we will see a difficulty with this soon

Try 2: use "square" of dependency graph:
connect nodes of dist 1 or 2

example:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\text{graph } G
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\text{graph } G^2
\end{array}
\]
degree of "square" graph:

\[ \text{deg} \leq \text{# nodes that can be reached in 1 or 2 steps in original graph} \]

\[ \leq d + d \cdot d \begin{array}{c} \uparrow \text{1 step} \\
\text{1st step} \leftrightarrow \text{2nd step} \\
\downarrow \text{2 steps} \end{array} \]

\[ \leq 2d^2 \]

why "square" graph?

1+3 both cause elts in a to be recolored

\[ \Rightarrow \text{step 2 needs to recolor 1, 2, 3 simultaneously} \]

For this lecture,

"Connected" component means

all nodes reachable in square graph
After 1st pass: orange $S_i$'s are "good", red $S_i$'s are "bad"

In pass 2, might need to fix neighbors of bad components:

Recall:

If $S_i$ not bad & has $\geq 2 \times l$ nodes in bad nbrs,
then $\geq 2 \times l$ nodes get recolored

Say $S_i$ "survives" if bad or has $\geq 2 \times l$ nodes in bad nbrs
We will show that connected components of "bad" sets $S_i$ are small: $O(\log n)$

Algorithm needs to recolor bad sets and possibly some of their nbrs in original graph (the ones that survive):

each bad set $S_i$ has $\leq d$ nbrs

$\Rightarrow$ total size ($\#s_i$'s) of component to recolor is $O(d\log n)$
How many repetitions of Pass 1?

\[ \forall S_i, \quad \Pr [S_i \text{ bad}] \leq 2 \cdot \sum_{i \leq \alpha n} \left( \frac{i}{2} \right) \leq 2 \cdot 2^\left( H(\alpha) - 1 \right) \leq \rho \]

Given dependency digraph \( G \), put edge between \( S_i \) and \( S_j \) if \( S_i \cap S_j \neq \emptyset \)

if \( S_{i_1}, S_{i_2}, \ldots, S_{i_m} \) are independent set

\[ (\text{so } S_{i_k} \cap S_{i_l} = \emptyset \quad \forall i_k, i_l) \]

then \( \Pr [S_{i_1} \ldots S_{i_m} \text{ all in } B] \leq \rho^m \)

since mutually independent
First try

Show no big component survives:

\[ \Pr[\text{specific big component survives}] \subseteq \Pr[\text{big independent set in component survives}] \leq p^{s'} \]

\[ \Pr[\text{any big component survives}] \leq \# \text{ potential big components in dependency graph} \cdot p^{s'} \]

What is a good bound? (\( \mathcal{S} \))? Way too big!!

Can use degree bound to improve!!

Can use degree bound to improve!!

How does \( S' \) compare to \( S \)?

If component is clique, then \( s' \) could be 1 but, use degree bound!
Plan: hope to show no big component survives.

if big component \( C \) survives.

then \( C \) has a big subtree

doesn't exist

then can find (less) big independent

Set in subtree

Well known fact:

\[ \text{# subtrees of size } u \text{ in graph of degree } \Delta \text{ is } \leq n \cdot \frac{1}{(\Delta - 1)(u+1)} (\Delta u)^u \]

\[ \text{# nodes } = n \]

\[ \leq n (e \Delta)^u \]

much much better than \( \binom{n}{u} \)

when \( \Delta \) is constant
Given a subtree of size $u$, it has an independent set of size $\geq \frac{u}{\Delta+1}$.

Why?

Repeat:

- Each round:
  - $|I|$ gets bigger by 1.
  - Subtree gets smaller by $\leq \Delta+1$.
- Remove $u$ and all neighbors of $u$ from subtree.

Until subtree is empty.

$\Rightarrow$ # rounds = $|I|$ $\geq \frac{u}{\Delta+1}$.
New try:

Show no big component survives:

\[ E \left[ \# \text{ of size } > S \text{ subtrees that survive} \right] \]
\[ \leq \sum_{i=S}^{m} E[\# \text{ size } i \text{ subtrees that survive}] \]
\[ \leq \sum_{i=S}^{m} (\# \text{ size } i \text{ subtrees}) \times \Pr[\text{size } i \text{ subtree survives}] \]
\[ \leq \sum_{i=S}^{m} m \cdot (ed)^i \times \left( \frac{i}{2^{d+1}} \right) \]
\[ \leq \sum_{i=S}^{m} m \cdot \left( \frac{1}{2^i} \right) \leq \frac{m}{2^{s-1}} \]

For \( s = \log 4m \)

Upper bound on expected \# of big components:

\[ \leq \frac{m}{4m} = \frac{1}{4} \]
By Markov's inequality:

\[ \Pr[\# \text{ of size } \geq \log 4m \text{ subtrees } > 0] < \frac{1}{4} \]

So \( \Pr[\# \text{ components of size } \geq \log 4m \text{ is } > 0] < \frac{1}{4} \)

\[ \Rightarrow \text{ expected # times to repeat first pass} \leq \frac{1}{4} \]
Polynomial Identity Testing

Is \( P(x) = (x+1)^2 \) the same as \( Q(x) = x^2 + 2x + 1 \)?

\[
\begin{align*}
\text{YES!} & \\
\text{What about } P(x) = (x+3)^3 (x-4)^3 \quad & \text{and } Q(x) = (x-4)^3 (x+3)^3
\end{align*}
\]

Obviously not! \( P(0) \neq Q(0) \).

Problem: given 2 polynomials \( P, Q \)

is \( P = Q \)?

i.e. is \( P(x) = Q(x) \forall x \)?

Problem': given polynomial \( R \)

is \( R = 0 \)?

i.e. is \( R(x) = 0 \forall x \)?

\[
\begin{align*}
\text{Let } & R(x) = P(x) - Q(x) \\
\text{then } & R = 0 \iff P = Q
\end{align*}
\]
Fact: If \( R \neq 0 \) has degree \( \leq d \) then 
\( R \) has at most \( d \) roots (recall: a "root" is \( x \) st. \( R(x) = 0 \))

Algorithm for deciding whether \( R = 0 \):

pick \( d+1 \) distinct inputs \( x_1 \ldots x_{d+1} \)

if \( \forall i \ R(x_i) = 0 \) output "\( R = 0 \)"
else \( \exists i \) st. \( R(x_i) \neq 0 \) output "\( R \neq 0 \)"

Runtime: \( O(d) \) evaluations of \( R \)