6.842 lecture 3

The Lovász Local Lemma

The Lovász Local Lemma

Another way to argue that it's possible that "nothing bad happens"

If $A_1, A_2 \ldots A_n$ are <u>bad</u> <u>events</u> how do we know that there is a positive probability that <u>none</u> occur?

$\Pr[A_i] \neq 1 \ \forall i$

if A_i's independent + "nontrivial":

$$\Pr[\cup A_i] \le 1 - \Pr[\cap \bar{A}_i]$$

$$= 1 - \prod \underbrace{\Pr(\bar{A}_i)}_{>0}$$

$$< 1$$

else, usual way: Union Bound

no assumptions on A_i's with respect to independence

$$\Pr[\cup A_i] \le \sum \Pr[A_i]$$

if each A_i occurs with prob $\le p$, then need $p < \frac{1}{n}$ to get interesting bound i.e. $\Pr[\cup A_i] < 1$

What if A_i's have "some" independence?

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

$$\forall \; J \subseteq [k], \; J \neq \emptyset \quad \text{then} \quad \Pr\Big[A \cap \bigcap_{j \in J} B_j\Big] = \Pr[A] \cdot \Pr\Big[\bigcap_{j \in J} B_j\Big]$$

note: $[k]$ means $\{1 \dots k\}$

def. $A_1 \dots A_n$ events

$D = (V, E)$ with $V = [n]$ is

"dependency digraph of $A_1 \dots A_n$"

if each A_i independent of all A_j that are not neighbors in D (i.e. all A_j s.t. $(i,j) \notin E$)

Lovász Local Lemma (symmetric version)

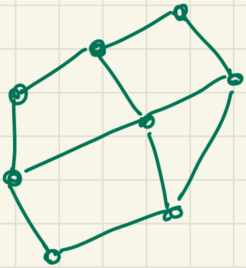
$A_1 \ldots A_n$ events s.t. $\Pr(A_i) \le p \ \forall i$

with dependency digraph D s.t. D

has max degree $\le d$.

If $epd \le 1$ then

$$\Pr\left[\bigcap_{i=1}^{n} \overline{A}_i\right] > 0$$



Union bnd	LLL
need	if $d \le 4$
$p < \frac{1}{n}$	only need
	$p \le \frac{1}{e\cdot(4+1)}$

Application

Thm. Given $S_1 \dots S_m \subseteq X$ $\quad |S_i| = \ell$

each S_i intersects at most d other S_j's

if $e\cdot(d+1) \le 2^{\ell-1}$

then can 2-color X such that

each S_i not monochromatic

previously needed $m < 2^{\ell-1}$

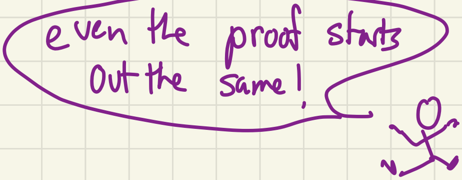
now no restriction on m

but there is a restriction on "degree"

ie. $\mathcal{H}$ is hypergraph with m edges each containing ℓ nodes + each intersecting $\le d$ other edges

Pf color each elt of X red/blue iid with prob $\frac{1}{2}$

$A_i \equiv$ event that S_i is monochromatic



$p = \Pr[A_i] = 1/2^{\ell-1}$

A_i indep of all A_j s.t. $S_i \cap S_j = \phi$

so depends on $\le d$ other A_j

Since $e \cdot p \cdot (d+1) = e \cdot \frac{1}{2^{\ell-1}} \cdot d+1 \leq 1$ ↖ by assumption

LLL $\Rightarrow \exists$ 2-coloring

∎

Comparison:

$\#\text{edges} = m$

size of edges $= \ell$

$m < 2^{\ell-1}$

$\#\text{edges} = m$

size of edges $= \ell$

each edge intersects with $\leq d$ others

$d+1 \leq \frac{2^\ell}{e}$

no dependence on m

Application 2: Boolean Formulae

Given CNF formula s.t. ℓ vars in each clause + each clause intersects $\le d$ other clauses

If $\frac{e(d+1)}{2^\ell} \le 1$ there is a satisfying assignment.

(n = # vars, m = # clauses)

How do you find a solution?

partial history:

Lovász	1975	nonconstructive (no fast algorithm to find soln)	$d \le 2^\ell/e$
Beck	1991	randomized algorithm but for more restrictive conditions on parameters	$d \le 2^{\ell/1000}$
Alon	1991	parallel version	$d \le 2^{\ell/8}$
	⋮		⋮

Moser 2009

negligible restrictions
for SAT
+ most other problems

$d \le \frac{2^\ell}{c}$

Moser Tardos

$\vdots$

Today Given $\varphi = \bigwedge_{i=1}^{m} C_i$ where C_i is clause with ℓ literals + each C_i intersects $\le d$ other C_j's.

If $d+1 \le \frac{2^\ell}{2^{2.1}}$ then can find of X s.t. each C_i satisfied in time poly in m, d, n

$\uparrow$ # variables

n = # vars
m = # clauses
d = max degree

$p = \Pr[\text{bad event } C_i \text{ unsat}] = 2^{-\ell}$

Moser's Algorithm:

```
1. Pick random assignment to vars 1..n
2. For each $i \in [m]$
        If $C_i$ unsatisfied
            Fix ($C_i$)          (*)
3. Output assignment
```

Fix (C):

```
rerandomize vars in $C$
    For $C'$ in $\{C\} \cup \{\text{nbrs of } C\}$
        If $C'$ unsatisfied then Fix ($C'$)
```

- how long does it run?
- does it terminate?

Observation

if terminates, we have a sat assignment

Why does it terminate?

idea view Moser's algorithm as a "compression algorithm"

- Input is random string R s.t. $|R| = t$
- Rule: if algorithm terminates or run out of bits in R, stop

  When stop, output E = encoding of trace of computation

Trace of Computation

do not write down bits of R that we <u>used</u>

- bit string b_i for $i \in [m]$

  $b_i = \begin{cases} 1 & \text{if Fix}(C_i) \text{ called on line } * \text{ (top level only)} \\ 0 & \text{o.w.} \end{cases}$

- for each recursive call to Fix

  save by writing which nbr, not full clause name

  record: (1) which nbr clause called

  (2) bits for recursive structure

  $b_1 = \begin{cases} 1 & \text{if } \exists \text{ child call} \\ 0 & \text{o.w.} \end{cases}$

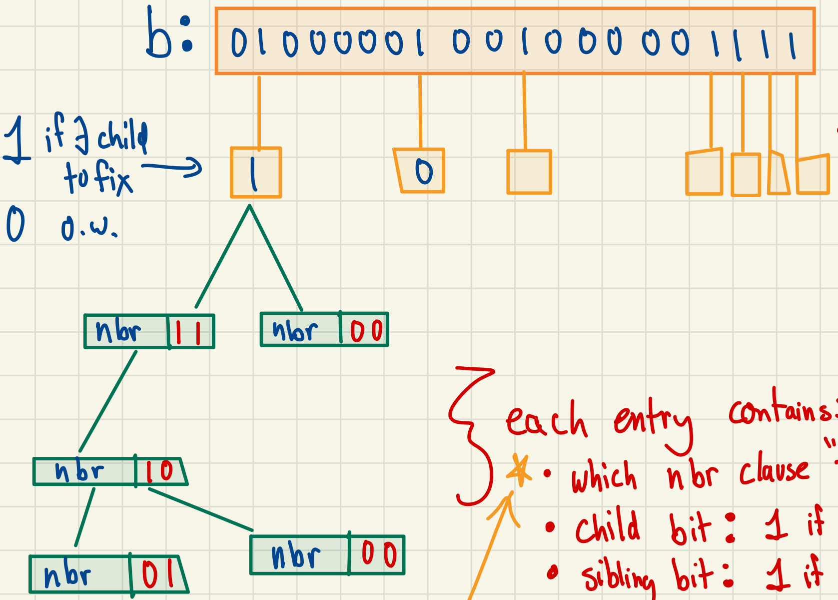
  $b_2 = \begin{cases} 1 & \text{if } \exists \text{ right sibling} \\ 0 & \text{o.w.} \end{cases}$

  (see picture)

- Final variable assignment
- Any remaining random bits in R

Warning: for this lecture will interchange T/F + 1/0

Picture of encoding



1 if ∃ child to fix

0 o.w.

for each "1" bit, draw tree of recursive DFS calls

each entry contains:

- which nbr clause "fix" called on : $\log_2(d+1)$ bits
- child bit: 1 if ∃ child to fix, 0 o.w.
- sibling bit: 1 if ∃ sibling to fix, 0 o.w.

why +1? because you can pick all F's again

Save since only write which nbr, not full clause ID.

Claim can recover R from encoding containing:

- b
- hanging trees
- final var assignment
- remaining unused bits of R

(Concatenated as bit strings)

example

$$R = \underbrace{\overset{0}{F}\,\overset{0}{F}\,\overset{0}{F}\,\overset{0}{F}\,\overset{0}{F}\,\overset{1}{T}\,\overset{1}{T}\,\overset{0}{F}\,\overset{0}{F}}_{\text{use for initial assignment}}\;\underbrace{\overset{1}{T}\,\overset{1}{T}\,\overset{0}{F}}_{\text{Fix 1}}\;\underbrace{\overset{0}{F}\,\overset{0}{F}\,\overset{0}{F}}_{\text{Fix 2}}\;\underbrace{\overset{1}{T}\,\overset{0}{F}\,\overset{1}{T}}_{\text{Fix 1}}\;\underbrace{\overset{1}{T}\,\overset{1}{T}\,\overset{0}{F}}_{\text{Fix 3}}\;\underbrace{\overset{11001}{\cdots}}_{\text{unused vars}}$$

$$\overset{①}{(x_1 \vee x_2 \vee x_3)}\;\overset{②}{(\bar{x}_1 \vee \bar{x}_2 \vee x_5)}\;\overset{③}{(x_4 \vee x_8 \vee x_9)}\;\overset{④}{(x_1 \vee x_6 \vee x_7)}$$

	①	②	③	④
Initially	(F F F)	(T T F)	(F F F)	(F T T)
Fix(1)	(T T F)	(F F F)	(F F F)	(T T T)
Fix(2)	(F F F)	(T T F)	(F F F)	(F T T)
Fix(1)	(T F T)	(F T F)	(F F F)	(T T T)
Fix(3)	(T F T)	(F T F)	(T T F)	(T T T)

Run of algorithm:

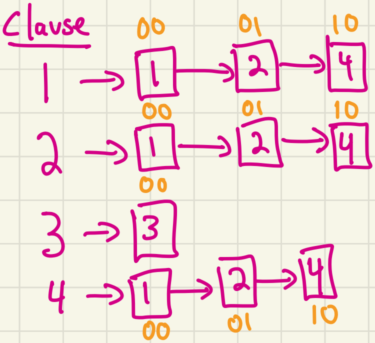
- Call Fix(1): use next bits of R (namely $\overset{x_1 x_2 x_3}{TTF}$)
  1 is now ok, 2 is now bad, 4 is still ok
- Recursively call Fix(2): next bits of R are $\overset{x_1 x_2 x_5}{FFF}$
  1 is now bad, 2 is now ok, 4 still ok
- Recursively call Fix(1): next bits of R are $\overset{x_1 x_2 x_3}{TFT}$
  1,2,4 are all ok
- Call Fix(3): next bits of R are $\overset{x_4 x_8 x_9}{TTF}$
  3 ok

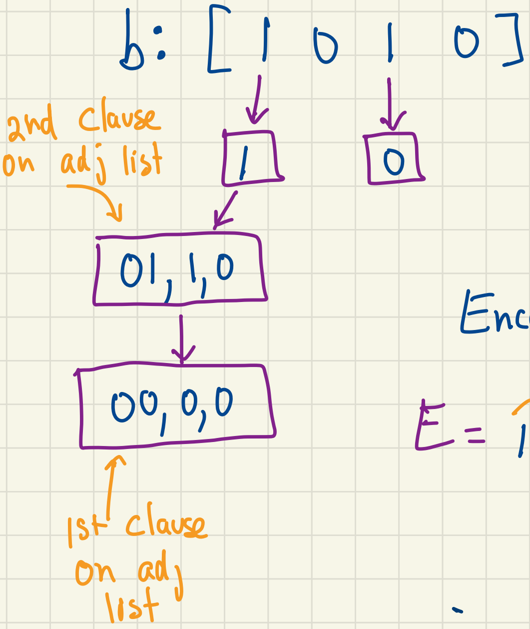
$$R = \overbrace{00000\ 11\ 00}^{\text{init assignment}}\overbrace{110}\overbrace{000}\overbrace{101}\overbrace{110}\ \overbrace{11001}^{\text{unused}}$$

encoding of example:

this is not part of encoding. Can be reconstructed by viewing input.

Adjacency list representation of clauses that intersect: (clauses intersect with themselves)





Encoding:

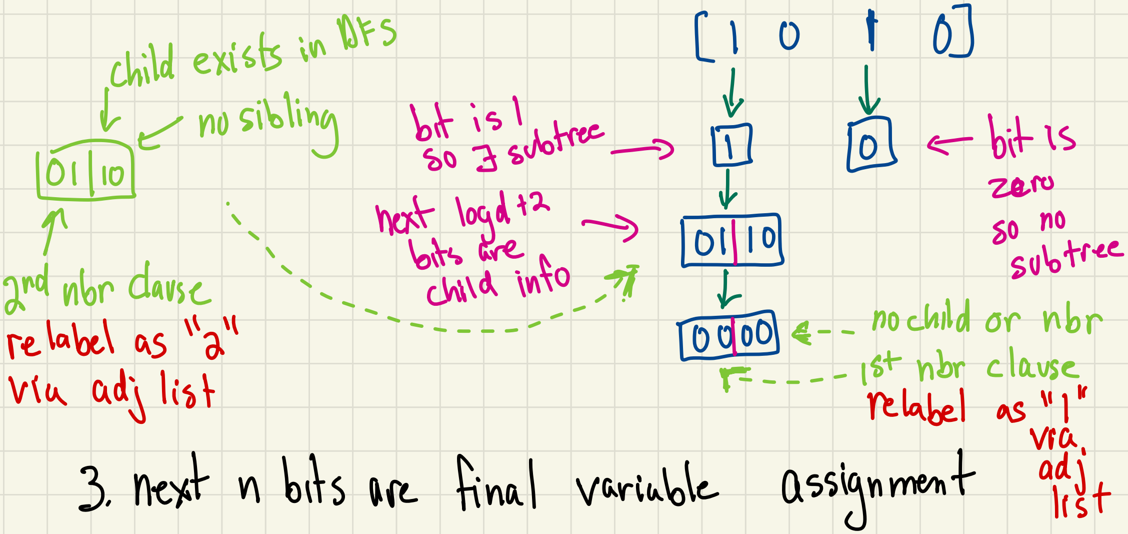
$$E = \overbrace{1010}^{b}\ \overbrace{1\ 0110\ 0000}^{\text{tree 1}}\ \overbrace{0}^{\text{tree 2}}\ \overbrace{1\ 0110\ 1110}^{\text{final variable settings}}\ \overbrace{11\ 001}^{\text{unused}}$$

How can we reconstruct R from encoding E?

First pass: Parse bits of encoding

1. 1st m bits give us b   [1 0 1 0]

2. Each "1" in b gives us hanging tree
   - recover structure of tree via DFS bits



3. next n bits are final variable assignment

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
T	F	T	T	F	T	T	T	F

4. remaining bits are unused bits

2nd Pass: Fill in R

- we already know what the unused bits are
- start with final var assignment

$X_1 = T$
$X_2 = F$
$X_3 = T$
$X_4 = T \leftarrow F$
$X_5 = F$
$X_6 = T$
$X_7 = T$
$X_8 = T \leftarrow F$
$X_9 = F \leftarrow F$

using tree:
look at calls to "fix" in reverse order

last call to Fix is Fix (3)
X_4, X_8, X_9 all False
⇒ last 3 bits of this part of R = TTF

$X_1 = T \leftarrow F$
$X_2 = F \leftarrow F$
$X_3 = T \leftarrow F$
$X_4 = T \leftarrow F$
$X_5 = F$
$X_6 = T$
$X_7 = T$
$X_8 = T \leftarrow F$
$X_9 = F \leftarrow F$

2nd to last call to Fix is Fix (1)
X_1, X_2, X_3 all False

⇒ 2nd to last 3 bits of this part of R = TFT

⇒ Initial vars: FFF F F TT FF

rolls: TTF FFF TFT TTF

$x_1 = T \leftarrow F \leftarrow T \leftarrow F$
$x_2 = F \leftarrow F \leftarrow T \leftarrow F$
$x_3 = T \leftarrow F \longleftarrow F$
$x_4 = T \leftarrow F$
$x_5 = F \longleftarrow F$
$x_6 = T$
$x_7 = T$
$x_8 = T \leftarrow F$
$x_9 = F \leftarrow F$

3rd to last call to Fix is
Fix(2) $\bar{x}_1, \bar{x}_2, x_5$ false
so $x_1 = x_2 = T$
$x_5 = F$
⇒ 3rd to last 3 bits of
this part of R = FFF

4th to last (first) call to Fix
is Fix(1) x_1, x_2, x_3 false

⇒ 4th to last (first) 3 bits
of this part of R = TTF

Initial assignment: (read rightmost setting)

$x_1 = T \leftarrow F \leftarrow T \leftarrow F$
$x_2 = F \leftarrow F \leftarrow T \leftarrow F$
$x_3 = T \leftarrow F \longleftarrow F$
$x_4 = T \leftarrow F$
$x_5 = F \longleftarrow F$
$x_6 = T$
$x_7 = T$
$x_8 = T \leftarrow F$
$x_9 = F \leftarrow F$

Initial assignment
FFF F F T T FF

$\underbrace{\text{FFF FF TT FF}}_{\text{initial}}\ \underbrace{\text{TTF}}_{\text{fix 1}}\ \underbrace{\text{FFF}}_{\text{fix 2}}\ \underbrace{\text{TFT}}_{\text{fix 1}}\ \underbrace{\text{TTF}}_{\text{fix 3}}$

$R = \overbrace{000\,00\,1100}\ \overbrace{110}\ \overbrace{000}\ \overbrace{101}\ \overbrace{110}\ \overbrace{11001}$

How compressed is this encoding of random bits?

Let w = # bits actually used by algo

s = # calls to Fix (including recursive)

Then $w = n + s \cdot \ell$

Length of trace encoding E_R

$$\le \underbrace{m}_{\text{describe } b} + (\underbrace{\log(d+1)}_{\text{describe node in ``hanging tree''}} + \underset{\nwarrow \text{ 2 bits for DFS}}{2}) \times s + \underset{\nwarrow \text{ output assignment}}{n} + \underbrace{|R| - \overset{\overset{n+s\cdot\ell}{\downarrow}}{w}}_{\text{Remaining}}$$

$$\le m + (\log(d+1) + 2 - \ell) \times s + \cancel{n - n} + |R|$$

So

$$|E_R| - |R| \le m + (\underbrace{\log(d+1)}_{} + 2 - \ell) \times s$$

we assumed

$d+1 \le 2^{\ell - 2.1}$

$\Rightarrow \log d+1 \le \ell - 2.1$

$$\le m + (\ell - 2.1 + 2 - \ell) \times s = m - 0.1 \times s$$

So, when s is big enough, $|\text{Trace}| \ll |R|$

Is compression of Trace a problem?

- we just gave a "lossless" compression scheme for random strings that lead to long runtimes.
- how many (random) strings can be compressed by b bits? $\leq 2^{-b}$ fraction

why? let compression fctn $f : \{0,1\}^t \to \{0,1\}^*$

f is 1-1

(can reconstruct x from f(x))

$$|\{0,1\}^t| = 2^t$$

$$\left|\bigcup_{t' < t-b} \{0,1\}^{t'}\right| < 2^{t-b}$$

fraction compressed by b bits $\leq \frac{2^{t-b}}{2^t} = 2^{-b}$

$O(m)$ bound on $\underbrace{\#\text{ calls to fix}}_{=s}$:

Suppose $s \geq 10(m+b)$

then $|E_R| - |R| \leq m - 0.1 \times s$

$$= m - 0.1(10(m+b))$$

$$= \cancel{m} - \cancel{m} - b$$

above $\Rightarrow$ at most 2^{-b} fraction of strings R can have $s \geq 10(m+b)$

e.g. let $b = 10$

$$\Pr[\#\text{ calls to fix} \geq 10m+100] \leq 2^{-10}$$