Coping with Intractability : (Fast) Exponential algorithms

3SAT is NP-complete

(Vertex Cover, 3-coloring, Subset Sum, Hamilton-Path, . . .)

~~If~~ $NP \neq P$, don't expect efficient (polynomial time) algorithms for these in worst-case.

OTOH, many of these problems do have to be solved. eg. SAT solving

<u>Cope with intractability:</u>

- Approximation algorithms
- Average-case complexity / random instances
- Heuristics (can't prove guaranteed performance, but works well on real-world instances)
  - → Sat solving
- Faster than trivial algos.
  - → (Fast) exponential algorithms

$3SAT \notin P$ (if $P \neq NP$)

Actually, it's conjectured that 3SAT is much harder.

Exponential Time Hypothesis (ETH)

3SAT requires $2^{\alpha n}$ time to solve (on n-variable instances) for some $\alpha > 0$

(e.g. no $2^{\frac{n}{\log n}}$ time algorithm).

Naive brute force alg.: $2^n \text{poly}(n)$ time.
(try all 2^n assignments to n vars)

Such ~~brute~~ force (try all solns) approach applies to vertex cover, 3-coloring, etc.

Often there's some structure which allows one to save on vanilla brute-force

(e.g. pruning some branches, clever local search etc.)

<u>Why?</u> - $2^{n/2}$ vs. 2^n makes a difference

- Interesting algorithmic ideas.

<u>Ultimate dream</u>: (for a problem of interest like 3SAT)

- Algorithm with runtime c^n $(c>1)$
- Hardness of solving in $(c-\varepsilon)^n$ for any $\varepsilon > 0$

· ETH says such c exists for 3SAT
- We have no clue what it might be

Fine-grained complexity:
- Poly time vs not poly time – coarse grained classification.
- n^3 vs not subcubic time – fine grained classification

Today: Fast exponential algos for 3SAT (which beat 2^n).

3SAT: instance: , $x_1 x_2 \cdots x_n$ Boolean vars

3CNF formula: with m clauses.

$\Phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_4 \vee x_5) \wedge (x_3 \vee \bar{x}_2 \vee x_9)$
$\wedge \cdots$

(clauses have width 3)

Goal: Find a 0-1 assignment to the x_i's that makes formula true.

$x_i, \overline{x_i}$ – <u>literals</u>

Beating brute force:

Algo 1: <u>Branching algo:</u>

Idea: $\Phi = (x_1 \vee x_3 \vee \overline{x_5}) \wedge \Phi'$

Branch on $\left.\begin{array}{l} x_1 = 1 \\ x_3 = 1 \\ x_5 = 0 \end{array}\right\} \to$ solve resulting formula on $(n-1)$ vars

$\Phi'|_{x_1=1}$

$$\begin{tikzcd}[column sep=small] & \bullet \arrow[dl, "x_1=1"'] \arrow[d, "x_3=1"] \arrow[dr, "x_5=0"] & \\ \Phi'|_{x_1=1} & \Phi'_{x_3=1} & \Phi'|_{x_5=0} \end{tikzcd}$$

$T(n) \le 3T(n-1) + poly(n)$

$T(n) \le 3^n \, poly(n)$

(worse than brute force)

<u>Branching:</u>

$$\begin{tikzcd}[column sep=small] & \Phi \arrow[dl, "x_1=1"'] \arrow[d, "\begin{array}{l} x_1=0 \\ x_3=1 \end{array}"] \arrow[dr, "\begin{array}{l} x_1=0 \\ x_3=0 \\ x_5=0 \end{array}"] & \\ \Phi'|_{x_1=1} & \Phi'|_{\substack{x_1=0 \\ x_3=1}} & \Phi'|_{\substack{x_1=0 \\ x_3=0 \\ x_5=0}} \end{tikzcd}$$

$(n-1)$ $\quad$ $(n-2)$ $\quad$ $(n-3)$ vars

$$T(n) \le T(n-1) + T(n-2) + T(n-3) + \text{poly}(n)$$

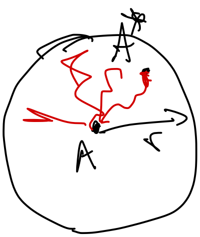
Solves to $T(n) \le (1.84)^n \, \text{poly}(n)$

LOCAL SEARCH

Suppose we knew an assignment A that is close to a satisfying assignment A^*

(in Hamming dist) (don't know this)

(Say A & A^* differ on r vars)



① Start with assignment A; $i=0$

② While $\exists$ at least one unsatisfied clause in A and $i \le r$

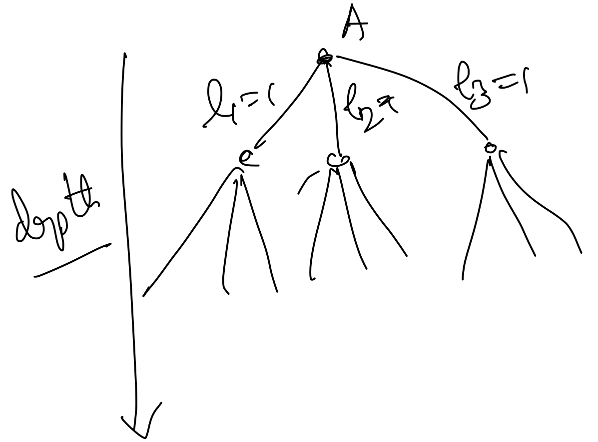
(a) Pick an arbitrary unsatisfied clause, say $\ell_1 \vee \ell_2 \vee \ell_3$

(b) $i{+}{+}$; Branch on each of the possibilities,

$A \leftarrow A|_{\ell_1 = 1}$

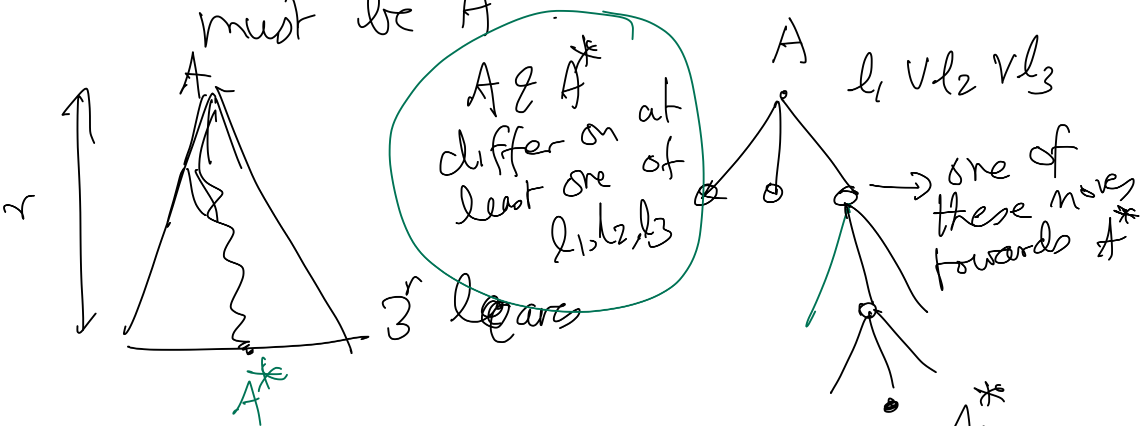
$A \leftarrow A|_{\ell_2 = 1}$

$A \leftarrow A|_{\ell_3 = 1}$



- Branching tree
- At each node, if you are a sat. assignment, output it & halt.

<u>Claim</u>. If algo didn't terminate before exploring to depth r, then one of the leaves must be A^*



Really just the first <u>silly</u> algo, but exploring only to depth r.

How do pick starting assignment A?

Try $A = 0^n$, and $A = 1^n$

One of these is within distance $\leq \frac{n}{2}$ from A^*

Runtime: $3^{\frac{n}{2}} \text{poly}(n) = (1.73)^n \text{poly}(n)$

Note: Picking random A also works & gives same guarantee.

Improvement:

Pick many more starting assignments A s.t the radius r can be taken smaller.

Picking $\frac{2^n}{\binom{n}{r}} \text{poly}(n)$ random values of A

will ensure A^* is within dist r of one of them.

Exercise: Prove this!

Runtime. $\frac{2^n}{\binom{n}{r}} \cdot 3^r \text{poly}(n)$

Optimize in r (details skipped):

$r = \frac{n}{4}$ is best choice.

Leads to $(1.5)^n \text{poly}(n)$ time.

Random walk algorithm (Schöning 1999)

Fact: Randomized $\underset{\text{poly}(n) \text{ time}}{\text{algo}}$ that succeeds with probability c^{-n} (say $\left(\frac{2}{3}\right)^n$)

$\Rightarrow$ By repeating it $c^n \text{poly}(n)$ times, we get an algo of runtime $c^n \text{poly}(n)$ that succeeds with high prob. $(1-2^{-n})$

Random walk algo:

① Pick a random initial assignment A

② While there is at least one unsatisfied clause in A & haven't run for $\boxed{> n}$ steps ($> 3n$)

a) Pick an arbitrary unsatisfied clause

b) Flip the value of a random variable in that clause.

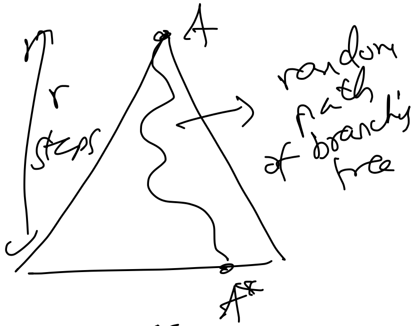
Prob[algo succeeds] $\geq$?

<u>Observation</u>. If $\text{dist}(A, A^*) = r$

A^* → some sat. assign

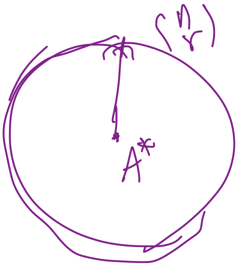
A → initial random assignment

then algo succeeds with prob $\geq \left(\frac{1}{3}\right)^r$



<u>Pf</u>: First r steps pick correct literal to flip (to go to A^*) with prob $\geq \left(\frac{1}{3}\right)^r$

$$\text{Prob[algo succeeds]} \geq \sum_{r=0}^{n} \underbrace{\binom{n}{r}\frac{1}{2^n}}_{\text{Prob}[\text{dist}(A, A^*) = r]} \cdot \underbrace{\left(\frac{1}{3}\right)^r}_{\substack{\text{lower bound on succ} \\ \text{prob if } \text{dist}(A, A^*) = r}}$$



$$= 2^{-n} \underbrace{\sum_{r=0}^{n} \binom{n}{r}\left(\frac{1}{3}\right)^r}$$

$$= 2^{-n}\left(1+\frac{1}{3}\right)^n = \frac{1}{2^n}\cdot\left(\frac{4}{3}\right)^n = \left(\frac{2}{3}\right)^n$$

$\Rightarrow$ Algo with runtime $\left(\frac{3}{2}\right)^n \text{poly}(n)$

An improved algorithm:

Small change: Run loop for $3n$ steps instead of n steps

Analysis: Instead of a beeline from A to A^*

analyze the chance of making at most r incorrect steps in the first $3r$ steps

(in this case algo will succeed!)

This prob. is $\geq$ Prob. of exactly r incorrect steps in the first $3r$ steps.

$$\geq \underbrace{\binom{3r}{r}\left(\frac{2}{3}\right)^r\left(\frac{1}{3}\right)^{2r}}$$

$$\Pr[\text{algo succeeds}] \geq \sum_{r=0}^{n}\binom{n}{r}2^{-n}\binom{3r}{r}\left(\frac{2}{3}\right)^r\left(\frac{1}{3}\right)^{2r}$$

Using Stirling's approx. for $\binom{3r}{r}$

$$\left(n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \right)$$

Above $\geq \Omega\left(\frac{1}{\sqrt{n}}\right) \cdot 2^{-n}\left(1+\frac{1}{2}\right)^n$

$$\geq \left(\frac{3}{4}\right)^n \Omega\left(\frac{1}{\sqrt{n}}\right)$$

Repeating this

Gives $\left(\frac{4}{3}\right)^n \text{poly}(n)$ time algo for 3SAT.

$(1.33)^n \text{poly}(n)$ — not bad!

Almost the best known runtime which is $\simeq (1.31)^n$

For k-SAT, Schöning random walk algo takes time $\left(2-\frac{2}{k}\right)^n \text{poly}(n)$ time.

Savings $\left(\frac{2}{k}\right)$ over brute force 2^n approaches 0 as k grows.

<u>Strong</u> exponential time hypothesis (SETH) asserts this is necessary.

[Origin of 3r:

- run for $r+2t$ steps
  with t incorrect steps.
- Write down the prob. of this
  and optimize in t, turns out $t=r$
  is best.]

$t=0$ was first analysis.]