Coping with Intractability: (Fast) Exponential algorithms

3SAT is NP-complete

(Vertex Cover, 3-color, 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. e.g. SAT solving

Cope with intractability:

- Approximation algorithms
- Average-case complexity/random instance
- Heuristics (can’t prove guaranteed performance, but work well on real-world instance)
  \rightarrow SAT solving
  
- Faster than trivial algo.
  \rightarrow (fast) exponential algorithms

3SAT \notin P (i.e. 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., $n = 2^{\log n}$ time algorithm).

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

Such brute force (try all solutions) 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.)

Why? $2^{n/2}$ vs. $2^n$ makes a difference

- Interesting algorithmic idea

Ultimate dream: (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 classification.
- $n^3$ vs not subcubic time — fine-grained classification.

---

Today: Fast exponential algo for 2SAT (which beat $2^n$).

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

2CNF formula: with $m$ clauses.

$\Phi = (x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_2 \lor \overline{x}_4 \lor \overline{x}_5) \land (x_3 \lor \overline{x}_2 \lor \overline{x}_9) \land \ldots$

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

(All clauses have width 3.)
\( x_i, \overline{x_i} \) - literals

**Beating brute force:**

**Alg 1: Branching also:**

IDEA: \( \overline{\Phi} = (x_1 \vee \overline{x_2} \vee \overline{x_5})) \land \overline{\Phi}' \)

Branch on \( x_1 = 1 \)

\( x_3 = 1 \)

\( x_5 = 0 \)

Resulting formula on \((n-1)\) vars

\[ T(n) \leq 3 T(n-1) + \text{poly}(n) \]

\[ T(n) \leq 3^n \text{ poly}(n) \]

(Worse than brute force)
\[ T(n) \leq T(n-1) + T(n-2) + T(n-3) + \text{poly}(n) \]

Solve for \( T(n) \leq (1.84) \text{poly}(n) \)

---

**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 \( A_j \) \( j = 0 \)
2. While \( \exists \) at least one unsatisfied clause in \( A \) and \( i \leq r \)
   a) Pick an arbitrary unsatisfied clause, say \( l_1 \lor l_2 \lor l_3 \)
   b) \( i+1 \) Branch on each of the possibilities,
      \[ A \leftarrow A_1 l_1 = 1 \]
      \[ A \leftarrow A_2 l_2 = 1 \]
      \[ A \leftarrow A_3 l_3 = 1 \]
- Branching tree
- At each node, if you are a cat's assignment, output it & halt.

Claim. If algo didn't terminate before exploring to depth \( r \), then one of the leaves must be \( A \).

Really just the first silly 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: $3^n \text{poly}(n) = (1.73)^n \text{poly}(n)$

Note: Picking random $A$ also works, but gives some guarantee.

**Improvement:**

Pick many more starting assignments $A$; 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 distance $r$ 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 \( (1.5)^n \poly(n) \) time.

---

Random walk algorithm (Schöning 1999)

**Fact**: Randomized algo that succeeds with probability \( c^{-n} \) (say \( (\frac{2}{3})^n \))

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

---

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 \( \geq n \) steps
   a) Pick an arbitrary unsatisfied clause
   b) Flip \( (> 3n) \)
b) Flip the value of a random variable in that clause.

\[ \text{Prob [also succeed]} \geq \frac{1}{2} \]

**Observation.** If \( \text{dist}(A, A^*) = r \), then also succeed with

\[ \text{Prob} \geq \left(\frac{1}{3}\right)^r \]

**Proof:** Fix \( r \) steps, pick correct literal to flip (to go to \( A^* \)) with

\[ \text{Prob [also succeed]} \geq \sum_{r=0}^{n} \binom{n}{r} \left(\frac{1}{2}\right)^r \cdot \left(\frac{1}{3}\right)^r \]

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

**Prob [dist(A, A^*) = r]**

Lower bound on success from \( \text{dist}(A, A^*) = r \).
$$= \left( \frac{2^n}{n} \left( 1 + \frac{1}{3} \right)^n = \frac{1}{2^n} \cdot \left( \frac{4}{3} \right)^n \approx \left( \frac{2}{3} \right)^n \right)$$

$\Rightarrow$ Algorithm 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 deadline from $A$ to $A^*$

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

(in this case also will succeed!)

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

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

$$\text{Pr}[\text{algo success}] \geq \sum_{r=0}^{n} \binom{n}{r} 2^{-n} \left( \frac{3r}{2} \left( \frac{2}{3} \right)^r \left( \frac{1}{3} \right)^{2r} \right)$$
Using Stirling's approx. for \((\binom{3^n}{n})\)

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

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

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

Repeating this gives \((\frac{4}{3})^n\) poly(\(n\)) time also for 3SAT.

\((\frac{4}{3})^n\) poly(\(n\)) — not bad!

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

For k-SAT, Schönig's random walk also takes \((2 - \frac{2}{k})^n\) poly(\(n\)) time.

Savings over brute force \(2^n\) approaches 0 as \(k\) grows.

Strong exponential time hypothesis (SETH) asserts this is necessary.
[Origin of Zr:
- Run for $r+2t$ steps with $t$ incorrect steps.
- Write down the prob. of this, and optimize in $t$, turns out $t = r$.
]

t=0 was first analysis. [}