“Problem X is NP-complete” – What does it mean?

Definition: X is in NP (=has efficient certificate system), and
any other problem from NP can be reduced to X in polynomial time
(with a Karp reduction a.k.a. many-one reduction)

Corollary: If X can be solved in polynomial time,
every problem from NP can be solved in polynomial time
(This is just a corollary, not an equivalent definition,
see Karp vs Turing reductions)

You can earn $1M if you solve X in polynomial time
(There are probably easier ways to earn $1M, e.g. become a football player)

“P = fast and NP-complete = slow” – Why so popular?

For natural problems, if in poly time, it is low deg poly

Computers will probably never run exp time algorithms in reasonable time

Many hard natural problems are NP-complete,
hence proving NP-completeness allows easy comparison with them

Clean definition of the class and reductions

Machine-independent, i.e. poly in Turing machine = poly in RAM model

Composable, i.e. poly(poly) = poly

“P = fast and NP-complete = slow” – Why too simplistic?

O(n^100) is poly time but not fast

O(n^2) is not fast if n=10^9

There may be faster-than-brute-force, subexponential time (e.g. 2^O(sqrt n)) algorithms for NP-complete problems

There may be fast approximation algorithms for NP-complete problems

Two ways out

FPT

Add more parameters!

“Maybe VC requires $\mathbf{2^{o(\mathit{n})}}$ time in general,
but you can solve it in $\mathbf{O(2^{\mathit{w}}\mathit{n})}$ time
on graphs with treewidth *w*”

Algorithmic techniques

FGC

Look closer at a parameter!

“You can solve LCS in $\mathbf{O(n^2)}$
but not in $\mathbf{O(n^{1.99})}$ time
(unless bad things happen)”

Connections between problems

“If SAT needs 2^n time, LCS needs n^2 time” – Conclusions?

Pessimistic: LCS needs n^2 time

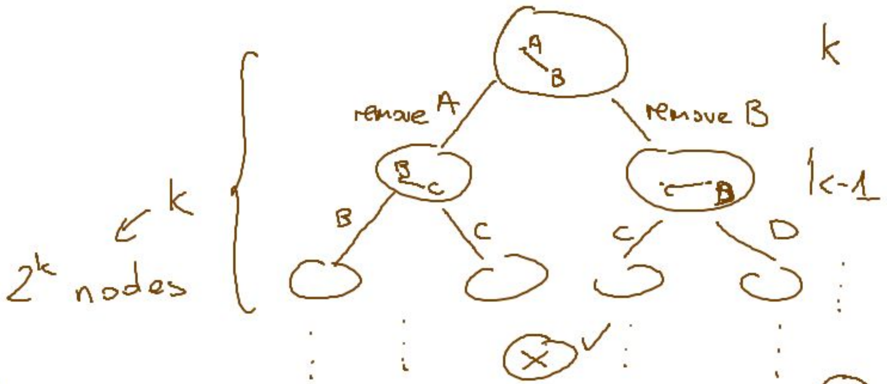
Optimistic: Let’s improve LCS, we’ll get an improvement for SAT for free

Realistic(?): Let’s first improve SAT, then we’ll work on LCS (it’s no easier)

Agnostic: SAT and LCS are related

BAR FIGHT PREVENTION

IN $\begin{cases} V = \{v_1, v_2, \ldots, v_n\} \\ E = \{\{v_1, v_3\}, \{v_3, v_7\}, \ldots\} \\ k \end{cases}$



OUT: Find a subset of V of size $\leq k$ st no enemies among remaining visitors.

2^n very naive 10^{301} $n=1000$, $k=10$

$\binom{n}{k}$ naive 10^{23}

$\binom{2k^2}{k}$+poly(n) kernel + naive 10^{16}

$2^k(n+m)$ bounded search tree 10^7

$m \leq n \cdot k$

R1 deg=0 → admit (KERNELIZATION

R2 deg>k → refuse

Apply rules exhaustively. Then $\forall_i$ deg_i $\in [1, k]$

Claim: $|E'| \leq k^2$ if YES-instance

R3 if $|E'| > k^2$, answer NO.

$\sum \text{deg} = 2|E'| \leq 2k^2 \rightarrow |V'| \leq 2k^2$

If a problem P admits an alg in $\underline{f}(k) \cdot poly(n)$, we say

$\binom{2k}{k} + poly(n)$

$2^k \cdot poly(n)$

P is fixed parameter tractable (with respect to parameter k)

If . . . . . . $n^{\underline{f}(k)}$

f has to be computable

slicewise polynomial (XP)

$\binom{n}{k} = O(n^k)$

$P \subseteq FPT \subseteq XP$

↑ informal

Problem: $\Sigma^* \to \{0,1\}$

Parameterized Problem: $\Sigma^* \times \mathbb{N} \to \{0,1\}$

Def: O*(f(k)) = O(f(k) poly(n))

E.g.: $3^k n^2$ is in $O^*(3^k)$

Kernelization algorithm

Input: (I, k)

Output: (I', k') or YES or NO

- polynomial time (in $|I|$)
- (I', k') is a YES-instance iff (I, k) is a YES-instance
- $|I'| + k' \leq f(k)$ // f has to be computable

Kernel → FPT

FPT → Kernel ? $f(k) \cdot n^c$ - algorithm

simulate it for n^{c+1} time

$f(k) > n$

We care about polynomial kernels $(|I'| + k' \leq \text{poly}(k))$