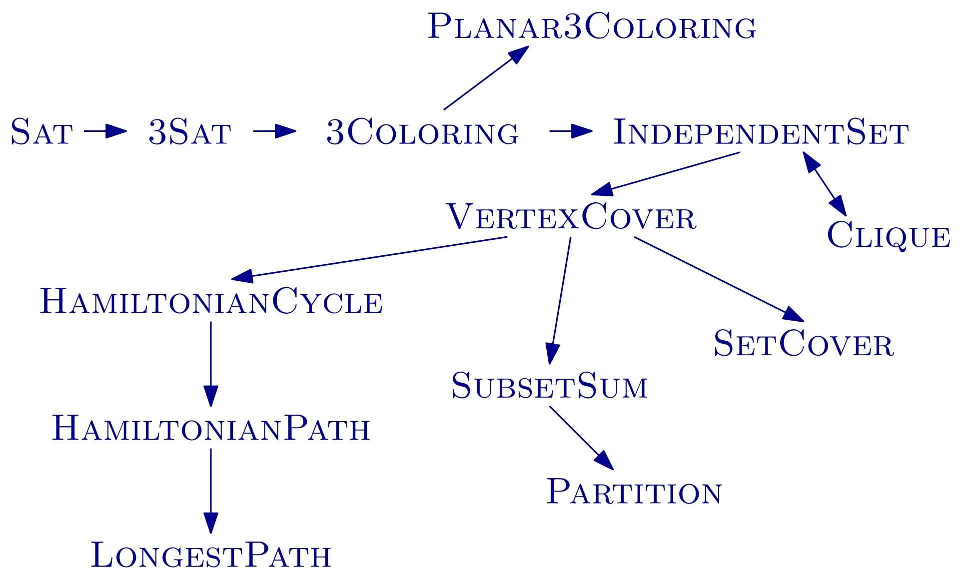
NP



What do all these problems have in common?

P and NP

For all of these problems the following is true: If the answer is "yes", then it is easy to show that this is indeed correct.

An algorithm $C(\cdot,\cdot)$ is a polynomial-time certifier for problem X if there is a polynomial $p(\cdot)$ such that for every instance I_X we have: the solution to I_X is "yes" if and only if

- there is a string t of length $|t| \leq p(|I_X|)$ such that
- $C(I_X, t)$ returns "yes", and
- $C(\cdot,\cdot)$ runs in polynomial time.

Note that if the answer to I_X is "no", there may be no certificate for this.

- Let P be the class of all problems that have a polynomial time algorithm.
- Let NP be the class of problems that have a polynomial-time certifier.

P and NP

Clearly we have $P \subseteq NP$.

So far, nobody has been able to find a problem that is in NP, but not in P.

So we do not know if $P \subsetneq NP$ or $P = NP$.

This is one of the millenium problems, and worth a million dollars.

Is it harder to find solutions than to check them?

If $P = NP$ then "creativity can be automated".

For instance, proofs for mathematical theorems can be found automatically.

P, NP, and EXP

Let EXP be the class of problems that have an exponential time algorithm. (Here, exponential time means $O(n^{p(n)})$, for some polynomial p).

For example $O(1.2^n)$, $O(2^n)$, $O(n!)$, $O(2^{n^3})$.

We have $P \subseteq NP \subseteq EXP$.

It is known that $P \neq EXP$.

Which of the two inclusions is proper?

NP-hardness and NP-completeness

A problem X is called NP-hard iff for every problem $Y \in$ NP we have

$$Y \le X.$$

In other words, X is harder than all problems in NP.

A problem X is called NP-complete iff

- $X \in$ NP, and
- X is NP-hard

In other words, X is one of the hardest problems in NP.

Cook-Levin Theorem: SAT is NP-complete.

Why is NP-completeness interesting?

Theorem: Let X be an NP-complete problem. Then X has a polynomial time algorithm if and only if $P = NP$.

Proof:

$\Leftarrow$: From $P = NP$ and $X \in NP$ follows that X has a polynomial-time algorithm.

$\Rightarrow$: Consider any $Y \in NP$.

We have $Y \le X$. Since X has a polynomial-time algorithm, this implies that Y has a polynomial-time algorithm.

Therefore $Y \in P$.

It follow $NP \subseteq P$, and therefore $NP = P$.

Why is NP-completeness interesting?

Theorem: Let X be an NP-complete problem. Then X has a polynomial time algorithm if and only if $P = NP$.

Finding a polynomial time algorithm for one NP-complete problem is equivalent to finding one for all problems in NP!

We believe $P \neq NP$, and so it is unlikely that an NP-complete problem has an efficient algorithm.

At least many smart people have failed to find an algorithm for these problems.

How to prove NP-completeness

To prove that a problem X is NP-complete, we need to

- show that $X \in NP$ (by giving a polynomial-time certifier for X)
- show that $Y \le X$ for some NP-hard problem Y.

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-Complete .

It's a surprisingly common phenomenon.

All the hard problems we studied are NP-complete, since there is a reduction from SAT.

Why prove NP-completeness?

When you encounter a problem for which you cannot find an efficient algorithm, you can prove that it is NP-complete.

- It shows to your boss/advisor that you are not too lazy to find a good algorithm.
- A paper with a heuristic or approximation algorithm is more likely to be accepted if you can show that the problem is hard.
- It makes you feel good.

Examples

Register allocation: Assign variables to (at most) k registers such that variables needed at the same time are not in the same register.

Interference graph: Nodes are variables, with an edge when variables are needed at the same time.

Register allocation is equivalent to coloring the graph with k colors.

$$\text{3COLORING} \leq k\text{-REGISTERALLOCATION},$$

for $k \geq 3$.

Examples

Class room scheduling: Given n classes and their meeting times, are k class rooms sufficient?

Again equivalent to k-COLORING.

Cellular telephone systems break up the frequency band into small bands. A cell phone tower gets one band. If towers are too close, they cannot get the same band.

The problem of assigning frequency bands to cell phone towers reduces to k-COLORING.

Subset Sum and KnapSack

Subset Sum Problem: Given n integers $a_1, a_2, ..., a_n$ and a target B, is there a subset of S of $\{a_1, ..., a_n\}$ such that the numbers in S add up precisely to B?

Knapsack: Given n items with item i having size s_i and profit p_i , a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?

$$\text{SUBSETSUM} \leq \text{KNAPSACK}$$

The input encoding matters

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise for you).

This implies that problem is hard only when (some of) numbers $a_1, a_2, ..., a_n$ are exponentially large compared to n.

The input encoding matters! When input is encoded in unary, the problem is in P .

Number problems of the above type are said to be weakly NP-Complete .

Lecture Planning

We want to plan a sequence of ℓ guest lectures. There are n possible speakers. In week i a subset L_i of these speakers is available.

Afterwards the students will do p projects about topics from the lectures. Project j requires at least one of the speakers from a subset P_j of the speakers.

Example: $\ell = 2$, $p = 3$, $n = 4$ speakers.

$L_1 = \{A, B, C\}$, $L_2 = \{A, D\}$.

$P_1 = \{B, C\}$, $P_2 = \{A, B, D\}$, $P_3 = \{C, D\}$.

LecturePlanning is clearly in NP. Is it also NP-hard?

NP-hardness proof

Reduction from 3Sat: For each variable x_i, make two lecturers x_i and $\bar{x}_i$. In week i, we can choose between them: $L_i = \{x_i, \bar{x}_i\}$.

Make a project for each clause!

Reduction from VertexCover: Given graph $G = (V, E)$ and $k > 0$, create a lecturer z_v for each node v.

Set $\ell = k$ and let $L_1 = L_2 = \ldots = L_k = \{z_v \mid v \in V\}$.

Make a project j for each edge $e_j = (v, w)$, where $P_j = \{z_v, z_w\}$.

More examples in the homework!