2nd part: Approximation Algorithms

Preamble: NP-Hardness

In the 30s we started to understand what is or isn't <u>effectively</u> computable

In the 70s we started to understand what is or isn't <u>efficiently</u> computable

In 1965 Edmonds defined what efficient means: an algorithm is "efficient" if its running time is $O(n^k)$ for some constant k (n = input size)

Problems for which a polynomial algorithm exists are called <u>tractable</u> → all the algorithms seen so far

If no polynomial algorithm exists the problem is said <u>intractable</u>

Examples:

1) Eulerian Circuit Problem: given an undirected graph, an Eulerian Circuit is a cycle that traverses all the edges only once.

This problem can be solved in linear time (exercise)

almost the same

2) Hamiltonian Circuit Problem: given an undirected graph, an Hamiltonian Circuit is a cycle that traverses all the vertices only once.

To date, no one knows a polynomial algorithm to solve it!

3) Minimum Spanning Trees: given a connected, undirected graph and a function $w: E \to \mathbb{R}$, output a spanning tree $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

almost the same

4) Traveling Salesperson Problem (TSP): given a complete undirected graph and a function $w: E \to \mathbb{R}$, output a tour (i.e. a cycle that visits every vertex exactly once) $T \subseteq E$ minimizing $\sum_{e \in T} w(e)$.

To date, no one knows a polynomial algorithm to solve it!

A much easier task: given a graph and a list of vertices C, <u>check</u> if C is an Hamiltonian circuit

Easy to solve : class P ("polynomial time")

1), 3) $\in P$

Easy to verify : class NP ("nondeterministic polynomial")

1), 3), 2), 4)

rookie mistake : NP $\neq$ not-polynomial

Decision problems :

to simplify the study of the complexity of problems, we limit our attention to the following class of problems:

<u>decision problems</u>: problems with a Boolean answer ? YES / NO

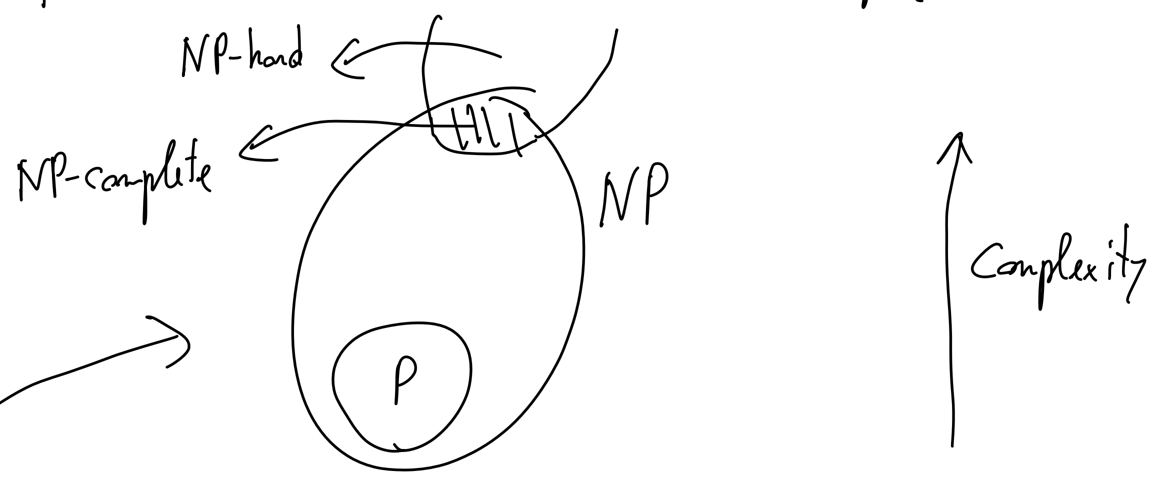
$\boxed{P}$ is the set of decision problems that can be solved in polynomial time

$\boxed{NP}$ is the set of decision problems with the following

property: if the answer is YES, then there is a proof of this fact that can be checked in polynomial time.
"certificate"

$\boxed{\text{coNP}}$ essentially the opposite of NP:

property: if the answer is NO, then there is a proof of this fact that can be checked in polynomial time.



NP-hardness: a computational problem is <u>NP-hard</u> if a polynomial-time algorithm for it would imply a polynomial-time algorithm for every problem in NP.

are the hardest problems in NP

A problem is NP-complete if it is both NP-hard and in NP

what we think the world would look like
today

the P vs NP question: $P \stackrel{?}{=} NP$

Why study NP-hardness:

- being NP-hard is <u>strong</u> <u>evidence</u> that a problem is intractable
- it suggests you should use a $\neq$ approach, such as
  - identify tractable special cases
  - compromise on correctness: $\longrightarrow$ <u>approximation</u> alg.

'71

Cook-Levin Theorem: 3SAT is NP-hard

<u>SAT</u>: formula satisfiability

AND NOT Clause

input: a Boolean formula like $(b \wedge \bar{c}) \vee (\bar{a} \wedge b)$

OR

output: it is possible to assign Boolean values to the variables $a, b, c, \dots$ so the the entire formula evaluates to TRUE?

A special case of SAT:

<u>3SAT</u>: a Boolean formula is in conjunctive normal form (CNF) if it is a conjunction (AND) of several clauses, each of which is the disjunction (OR) of several literals, each of which is either a variable or its negation

example: $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee d) \wedge (\bar{a} \vee c \vee d)$

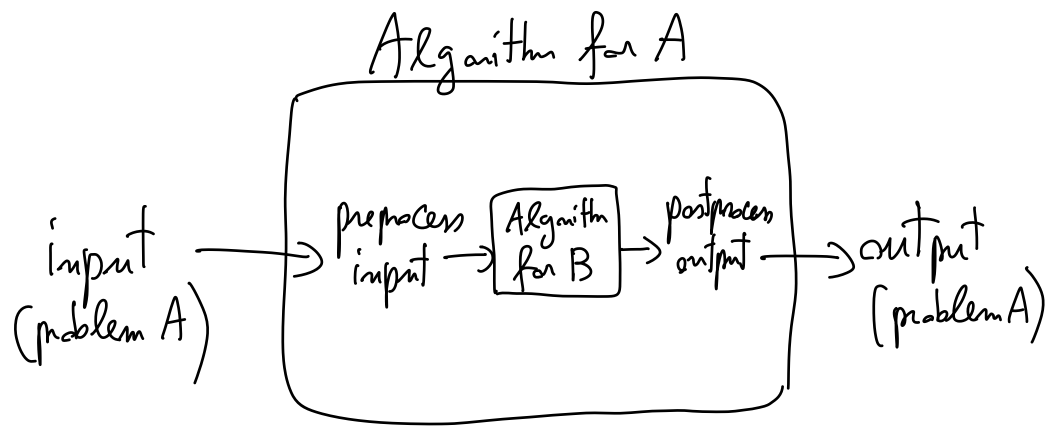
a 3-CNF formula is a CNF formula with exactly 3 literals per clause.

How to show a problem is NP-hard?

Reductions

To prove that a problem is NP-hard we use a reduction:

Reducing problem A to problem B means describing an algorithm to solve A under the assumption that an algorithm for B exists



$$A < B$$

"reduces to"

B is then as hard as A