NP-complete? NP-hard?

Some Foundations of Complexity

Prof. Sven Hartmann
Clausthal University of Technology
Department of Informatics

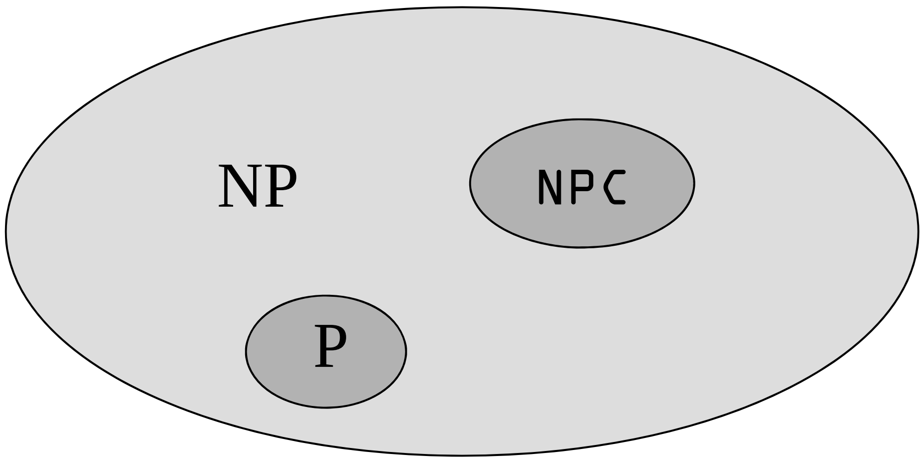
Tractability of Problems

- Some problems are undecidable:
  - no computer can solve them
  - Example: Turing's Halting Problem
- Other problems are decidable, but intractable:
  - When instances of the problem become large, we are unable to solve them in reasonable time
- What constitutes "reasonable time"?
- Usually we expect a polynomial-time algorithm
  - Given any instance of the problem with input size n (in some encoding)
  - Worst case running time is $O(n^c)$ for some constant c

The Classes P, NP and NPC

- There are three interesting classes of decision problems:
  - P = class of problems that can be solved in polynomial time
  - NP = class of problems that can be verified in polynomial time
  - NPC = class of problems in NP that are as hard as any problem in NP

- Most theoretical computer scientists have this picture in mind:



The Class NP

- NP = class of problems that can be verified in polynomial time
  - What does “verified” mean?

- A problem is verifiable in polynomial time:
  - given a certificate of a solution, the certificate can be shown to be correct in polynomial time

- Example: PATH (decision)
  - Given a graph G, nodes u and v, and an natural number k, decide whether there exists a path from u to v consisting of at most k edges
  - The certificate might be a path in the graph G
  - It is easy to check (in polynomial time) that the path goes from u to v, and has no more than k edges

Example: Hamiltonian Cycle

- Hamiltonian cycle
  - A simple path containing every node in a graph G
- HAM-CYCLE:
  - Given a graph G decide whether G contains a Hamiltonian cycle
- The naïve algorithm for solving HAM-CYCLE runs in $\Omega(m!)=\Omega(2^m)$ time, where *m* is the number of nodes
- However, as a certificate we can use an ordered sequence of *m* vertices
  - We can easily verify whether the sequence is indeed a Hamiltonian cycle

Reduction between Decision Problems

- Given two decision problems A and B
- Suppose we have a black-box solving problem A in polynomial time. Can we use the black-box to solve problem B in polynomial time?

- A polynomial-time reduction from A to B is a transformation that maps instances of A to instances of B such that:
  - The transformation takes polynomial time
  - The answer is the same (the answer for the instance of A is YES if and only if the answer for the instance of B is YES)

- We say that A is polynomial-time reducible to B if there exists a reduction of A to B
  - Often write $A \leq_p B$
  - If $A \leq_p B$, then B in class P implies that A is in class P

The Class NPC (= NP-Complete Problems)

- NPC = class of problems in NP that are as hard as any problem in NP
  - When is a problem in NP as hard as another problem in NP?

- A problem B in NP is as as hard as a problem A in NP:
  - B is polynomial-time reducible to A

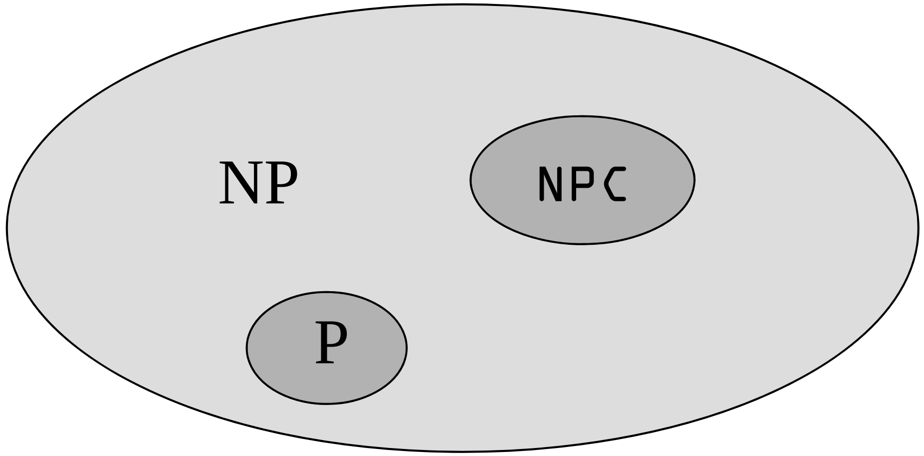
- So if a problem B is in NPC then:
  - Any other problem in NP can be polynomial-time reduced to it
  - If B can be solved in polynomial time then so can any other problem in NP

  - In this sense, problems in NPC are the “hardest” among the problems in NP

  - So far no polynomial-time algorithm is known for any problem in NPC
  - But so far we neither have a proof that a polynomial-time algorithm cannot exist for any problem in NPC

Relation among P, NP and NPC

- We know for sure:
  - $P \subseteq NP$
  - $NPC \subseteq NP$

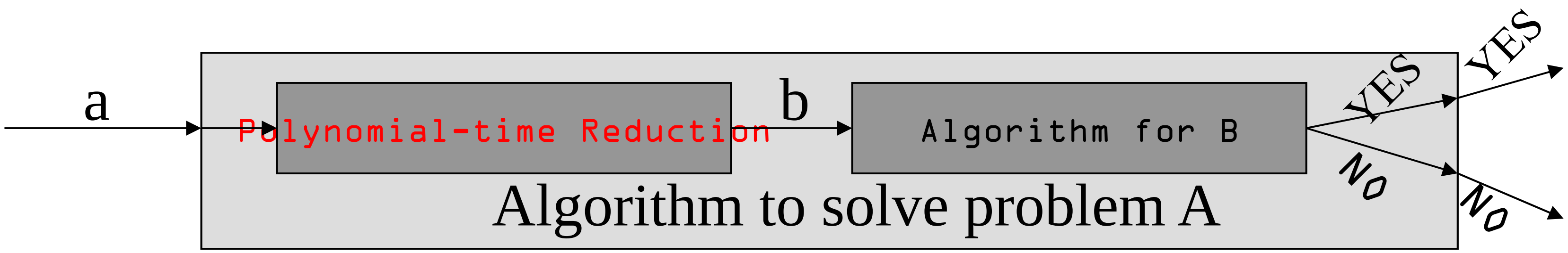


- But unclear whether:
  - $P = NP$ (or $P \subset NP$, or $P \neq NP$) ???
  - $NPC = NP$ (or $NPC \subset NP$, or $NPC \neq NP$) ???

- $P \neq NP$ is one of the deepest, most perplexing open research problems in (theoretical) computer science since 1971
  - One of the seven 1-million-dollar problems

Why reductions are of interest

- Let A and B be problems in NP such that A $\leq_p$ B
  - If the algorithm for B is polynomial-time, then so is the algorithm for A
    - Consequently, A is no harder than B (or B is no easier than A)
  - If A is in NPC, then so is B
- How to prove a problem B to be in NPC ??
  - prove B is in NP
  - Choose some problem A in NPC, and ...
  - ... establish a polynomial-time reduction of A to B



Why NPC is of interest

- If a problem is proved to be NPC, this is a good evidence for its intractability (or at least for its hardness)
- Might not want to waste time on trying to find an efficient algorithm
- Instead, focus on design approximate algorithm or heuristic or a solution for a special case of the problem
- Some problems looks very easy at the first glance, but are NPC

- Sometimes we can show that a problem is as hard as any problem in NP, but we are not able to show the problem is in NP
  - Such problems are called NP-hard
  - Hence, NP-complete = NP + NP-hard

Decision vs. Optimization Problems

- Decision problem:
  - solving the problem by giving an answer "YES" or "NO"

- Optimization problem:
  - solving the problem by finding the optimal solution

- Examples:
  - PATH (decision)
    - Given a graph G, nodes u and v, and an natural number k, decide whether there exists a path from u to v consisting of at most k edges
  - SHORTEST-PATH (optimization)
    - Given a graph G, and nodes u and v, find a path from u to v with as few edges as possible

Decision vs. Optimization Problems

- Decision problems are not harder than optimization problems
  - If there is an algorithm for an optimization problem, the algorithm can be used to solve the corresponding decision problem
  - Example: SHORTEST-PATH for PATH
  - If an optimization problem can be solved in polynomial-time, then so can the corresponding decision problem
- By definition, P, NP and NPC are classes of decision problems
- The discussion can be extended to optimization problems, though

The SAT Problem

- One of the first problems to be proved to be in NPC is *Satisfiability* (SAT):
  - Given a Boolean expression on *n* variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - **Cook's Theorem:** The satisfiability problem is NP-Complete
  - Cook's proof uses first principles, not yet reduction