Classes of Problems

- Polynomial Time Verification
- The Classes P and NP
- The Classes EXP and coNP
- NP-Hard and NP-Complete Problems
- Proving NP-Hardness
- A first NP-Complete Problem

Imdad ullah Khan

NP-Hard and NP-Complete Problems

A problem X is NP-Hard, if every problem in NP is polynomial time reducible to X

$$X \in \mathrm{NP} \quad \text{AND} \quad \forall\, Y \in \mathrm{NP}, \quad Y \leq_p X$$

A problem $X \in \mathrm{NP}$ is NP-Complete, if every problem in NP is polynomial time reducible to X

$$X \in \mathrm{NP} \quad \text{AND} \quad \forall\, Y \in \mathrm{NP}, \quad Y \leq_p X$$

These problems are at least as hard as any problem in NP

Let NPC be the (sub)class of NP-Complete problems

▷ It is the set of hardest problems in NP

If any NP-complete problem can be solved in poly time, then all problems in NP can be, and thus $\mathrm{P} = \mathrm{NP}$

How to prove NP-COMPLETEness

A problem X is NP-COMPLETE, if

1. $X \in \mathrm{NP}$
2. $\forall\ Y \in \mathrm{NP}\ Y \leq_p X$

How to prove a problem NP-COMPLETE ?

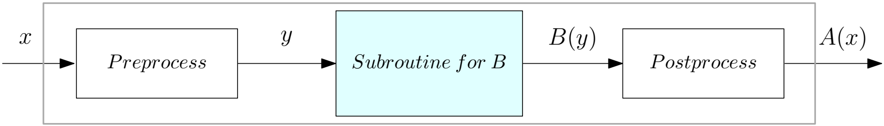
- Proving NP is relatively easy (in many cases)
- Can we do so many reductions?

Polynomial Time Reduction: Algorithm Design Paradigm

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem A can be solved using a polynomial amount of computation plus a polynomial number of calls to a solution of problem B

Subroutine for B takes an instance y of B and returns the solution $B(y)$



Algorithm for A transforms an instance x of A to an instance y of B. Then transforms $B(y)$ to $A(x)$

Suppose $A \leq_p B$.

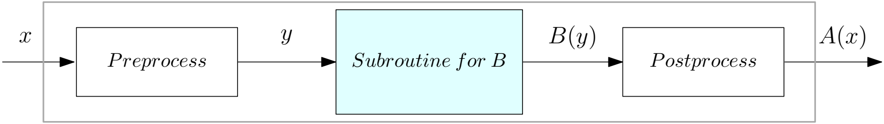
If B is polynomial time solvable, then A can be solved in polynomial time

Polynomial Time Reduction: Tool to Prove Hardness

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem A can be solved using a polynomial amount of computation plus a polynomial number of calls to a solution of problem B

Subroutine for B takes an instance y of B and returns the solution $B(y)$



Algorithm for A transforms an instance x of A to an instance y of B. Then transforms $B(y)$ to $A(x)$

Suppose $A \leq_p B$.

If A is NP-Complete, then B is NP-Complete

▷ **Why?**

Proving NP-COMPLETE Problems

A problem X is NP-COMPLETE, if

1. $X \in \mathrm{NP}$
2. $\forall\ Y \in \mathrm{NP}\ Y \leq_p X$

To prove X NP-COMPLETE, reduce an NP-COMPLETE problem Z to X

If Z is NP-COMPLETE, and

1. $X \in \mathrm{NP}$
2. $Z \leq_p X$

then X is NP-COMPLETE

1. $X \in \mathrm{NP}$ is explicitly proved
2. $\forall\ Y \in \mathrm{NP},\quad Y \leq_p X$ follows by transitivity

   $\forall\ Y \in \mathrm{NP},\quad Y \leq_p Z$ is true as Z is NP-COMPLETE

   $[Y \leq_p Z \ \wedge\ Z \leq_p X] \implies Y \leq_p X$

Proving NP-COMPLETE Problems

A problem X is NP-COMPLETE, if

1. $X \in \mathrm{NP}$
2. $\forall\ Y \in \mathrm{NP}\ Y \leq_p X$

How to prove a problem NP-COMPLETE?

- Proving NP is relatively easy
- Can we do so many reductions?

Template of proving problems to be NP-COMPLETE

We proved that	CLIQUE(G,k) is NP-COMPLETE
Suppose we have the theorem	CLIQUE(G,k) $\leq_p$ IND-SET(G,k)
Then we can conclude that	IND-SET(G,k) is NP-COMPLETE