Design and Analysis of Algorithms

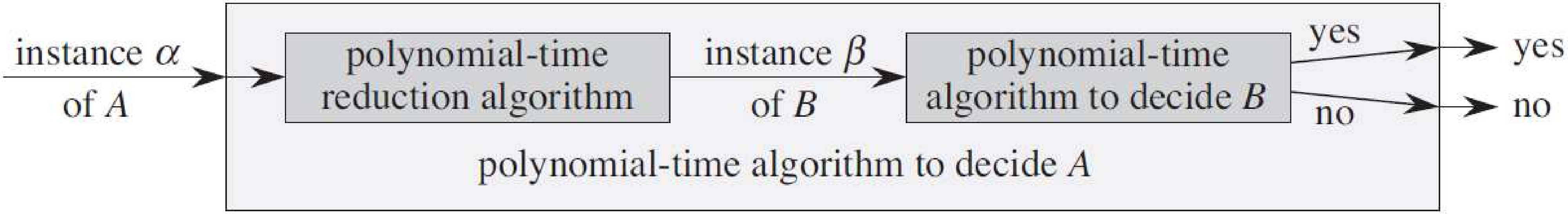
National University of Singapore

CS3230

Lecture 11
NP-Completeness

Warut Suksompong

Reductions



Suffices to show:

- Reduction runs in polynomial time
- If α is a YES-instance of A, β is a YES-instance of B
- If β is a YES-instance of B, α is a YES-instance of A

3-SAT

1-SAT is trivial: a formula can be satisfied iff no literal occurs simultaneously with its negation

YES-instance: $(\overline{x_1}) \wedge (x_2) \wedge (x_3) \wedge (\overline{x_4})$

NO-instance: $(\overline{x_1}) \wedge (x_2) \wedge (x_3) \wedge (x_1)$

SAT where each clause contains exactly 3 literals (not necessarily distinct)

2-SAT can be solved in polynomial time (not required to know for this Module)

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Satisfying assignment: $x_1 = \text{True}, x_2 = \text{True}, x_3 = \text{False}, x_4 = \text{True}$

Non-satisfying assignment: $x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{False}, x_4 = \text{False}$

Φ is a YES-instance if and only if it admits at least one satisfying assignment.

3-SAT $\leq_P$ INDEPENDENT-SET

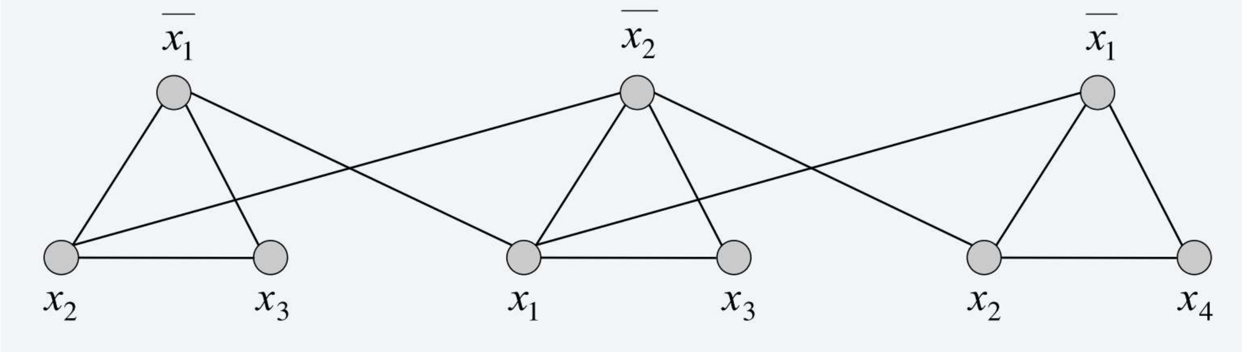
Given an instance Φ of 3-SAT, goal is to construct an instance (G, k) of INDEPENDENT-SET so that G has an independent set of size k iff Φ is satisfiable.

3-SAT $\leq_P$ INDEPENDENT-SET

Given an instance Φ of 3-SAT, goal is to construct an instance (G, k) of INDEPENDENT-SET so that G has an independent set of size k iff Φ is satisfiable.

Reduction

- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses

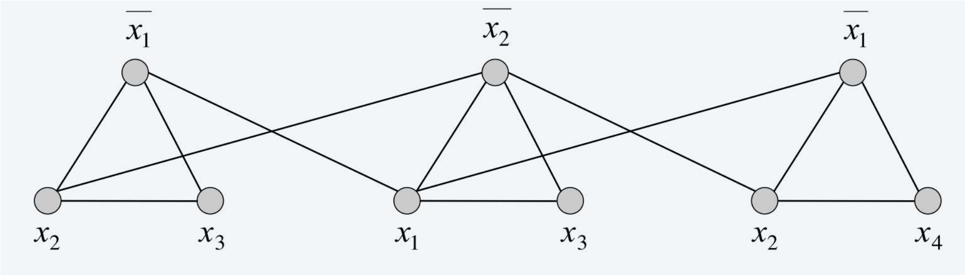


$$(\overline{x_1} \vee x_2 \vee x_3)$$
$$\wedge\ (x_1 \vee \overline{x_2} \vee x_3)$$
$$\wedge\ (\overline{x_1} \vee x_2 \vee x_4)$$

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction

- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses



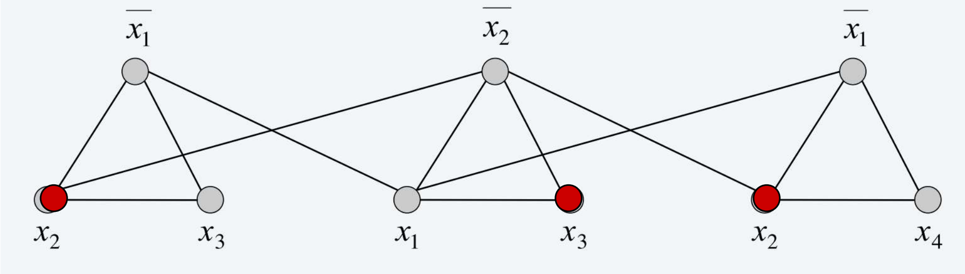
$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Reduction clearly runs in polynomial time.

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction

- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses



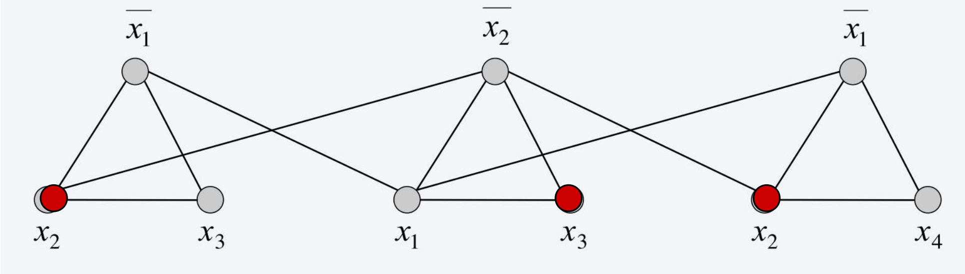
$(\overline{x_1} \vee x_2 \vee x_3)$
$\wedge (x_1 \vee \overline{x_2} \vee x_3)$
$\wedge (\overline{x_1} \vee x_2 \vee x_4)$

Suppose Φ is a YES-instance. Take any satisfying assignment for Φ and select a true literal from each clause. Corresponding k vertices form an independent set in G.

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction

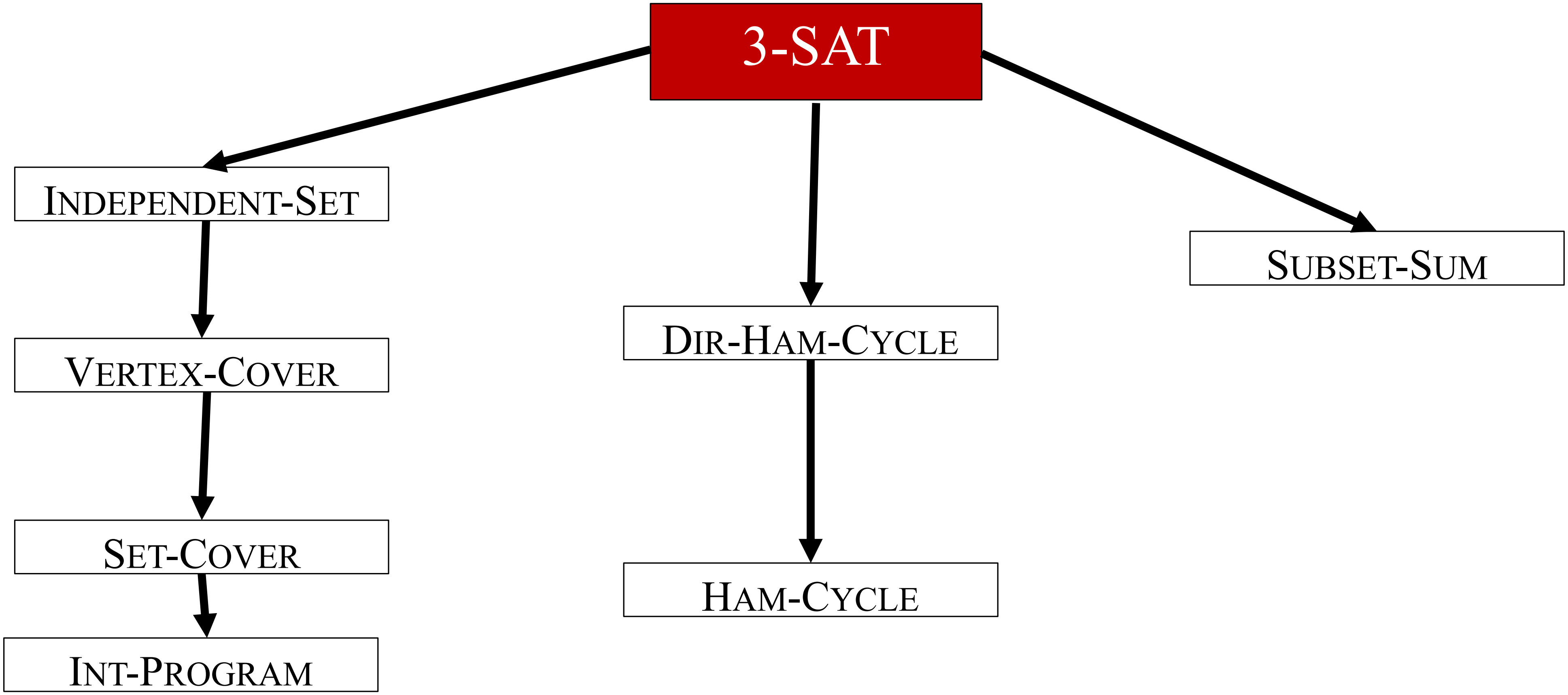
- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses



$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Suppose (G, k) is a YES-instance. Let S be the independent set of size k. Each of the k triangles must contain exactly one vertex in S. Set these literals to true, so all clauses satisfied.

Reductions


3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM

What have we shown?

If 3-SAT does not have a poly-time algorithm, then none of these problems have poly-time algorithms!

Why do we care?

- Problems believed to be hard are the basis of secure cryptosystems. For example, credit cards, https internet security, digital signatures, etc.

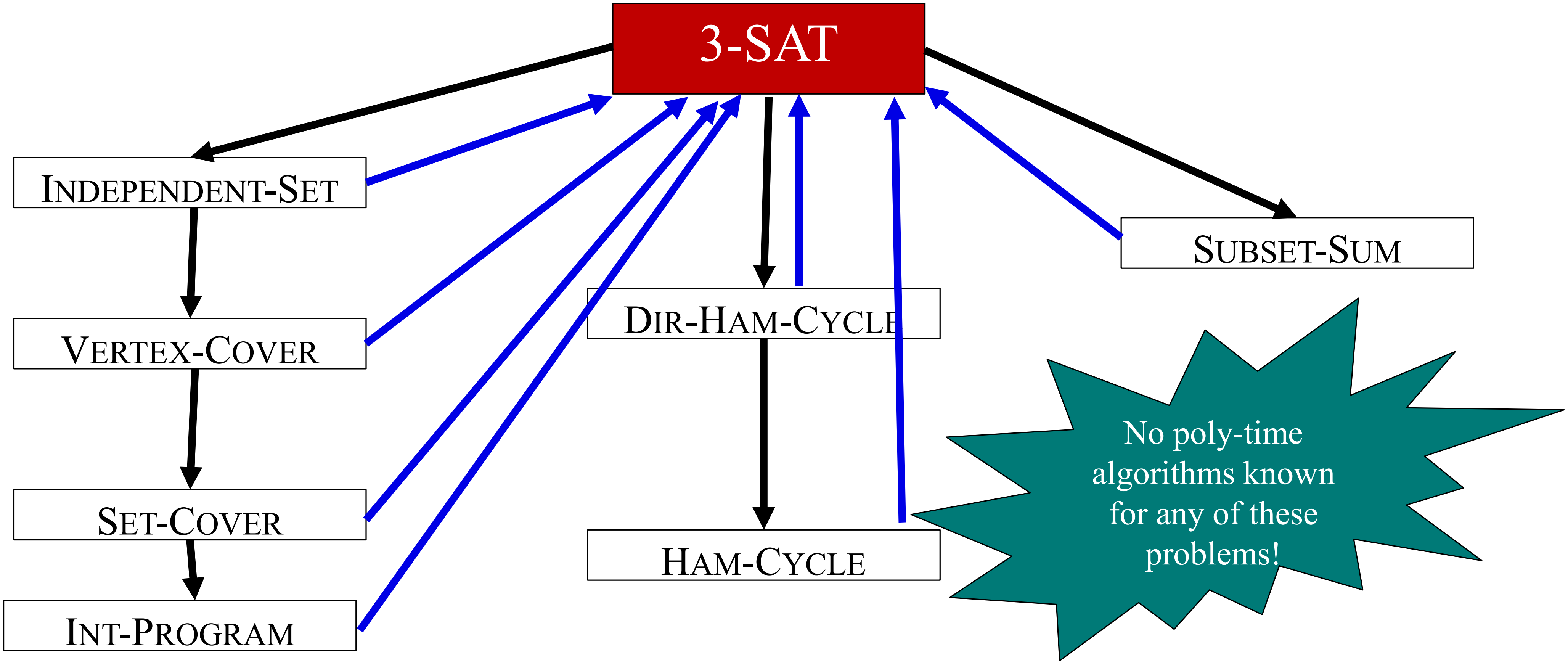
- Entire area of **complexity theory** devoted to comparing the hardness of different problems.

But…

If 3-SAT does not have a poly-time algorithm, then none of these problems have poly-time algorithms!

What if someone finds an $O(n^2)$-time algorithm for 3-SAT tomorrow? Seems dangerous to say that all these problems are hard based on lack of progress on this one specific problem…

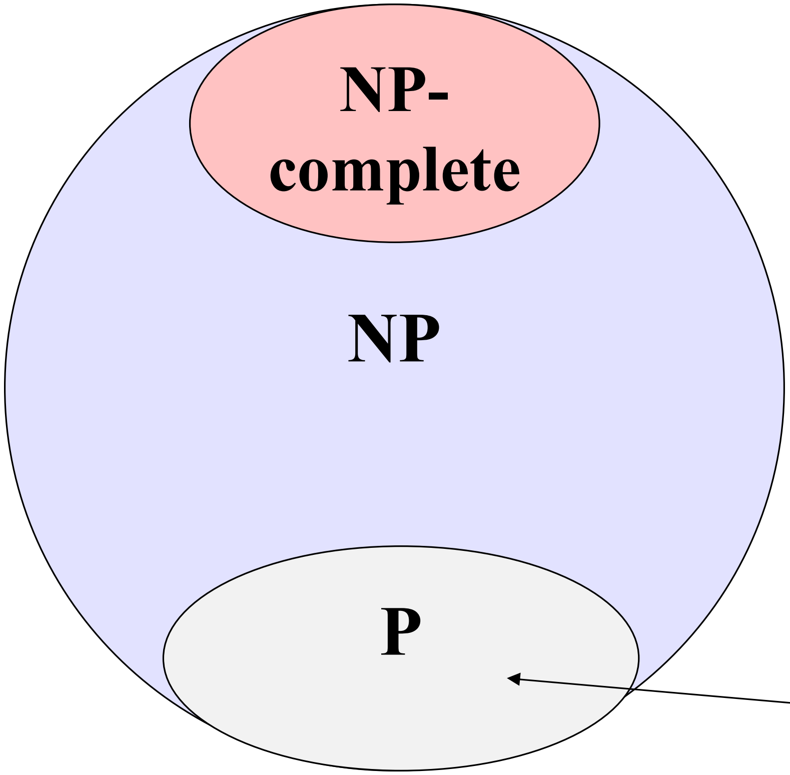
Reductions


3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM
No poly-time algorithms known for any of these problems!

In fact…

- Hundreds of problems, many encountered routinely in applications, are **computationally equivalent** to 3-SAT.
  - A poly-time algorithm for any of these problems implies poly-time algorithm for others.
  - Equivalently, if you believe any of these problems is hard, all of the problems are hard.

Complexity Classes



Problems solvable in (deterministic) poly time

The class NP

NP (short for “non-deterministic polynomial”) is the class of problems for which polynomial-time verifiable **certificates** of YES-instances exist.

Equivalently, this is the class of problems that can be solved in polynomial time by a “non-deterministic Turing machine”, hence the name “non-deterministic polynomial”.

Example: SUBSET-SUM

Recall: In SUBSET-SUM, given a list of integers S and target t, problem is to decide if there is $S' \subseteq S$ that sums up to t.

Certificate is the subset S' itself. Verifier checks whether the sum of elements of S' is t in polynomial time.

Hence, SUBSET-SUM is in NP.

$S = \{2, 4, 5, 7, 8, 10\}$

$t = 19$

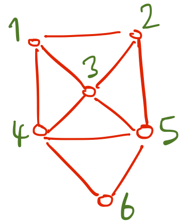
certificate: $S' = \{2, 7, 10\}$

Example: HAM-CYCLE

Recall: In Ham-Cycle, given a graph G, problem is to decide whether there is a simple cycle that visits each vertex once.

Certificate is the cycle itself. Verifier checks in polynomial time whether it is a cycle and visits each vertex once.

Hence, HAM-CYCLE is in NP.

$G =$ 

certificate : $1 \to 3 \to 2 \to 5 \to 6 \to 4 \to 1$

$\mathbf{P} \subseteq \mathbf{NP}$

Any problem in **P** is in **NP**. Why?

The certificate can be anything. The verifier $V(x,\cdot)$ can solve for the instance x by itself and check if it is a YES-instance.

Why NP?

NP captures natural search problems, similar to searching for a needle in a haystack. We know we have the needle when we see it!

> *"NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly."* — Christos Papadimitriou

NP-Hard and NP-Complete

A problem A is said to be **NP-Hard** if for every problem B in **NP**:

$$B \leq_P A$$

"at least as hard as every problem in NP"

Problem A is said to be **NP-Complete** if it is in **NP** and also **NP**-Hard.

NP-Hard and NP-Complete

A problem A is said to be **NP-Hard** if for every problem B in **NP**:

$$B \leq_P A$$

Problem A is said to be **NP-Complete** if it is in **NP** and also **NP**-Hard.

The hardest problems in NP

The Cook-Levin Theorem

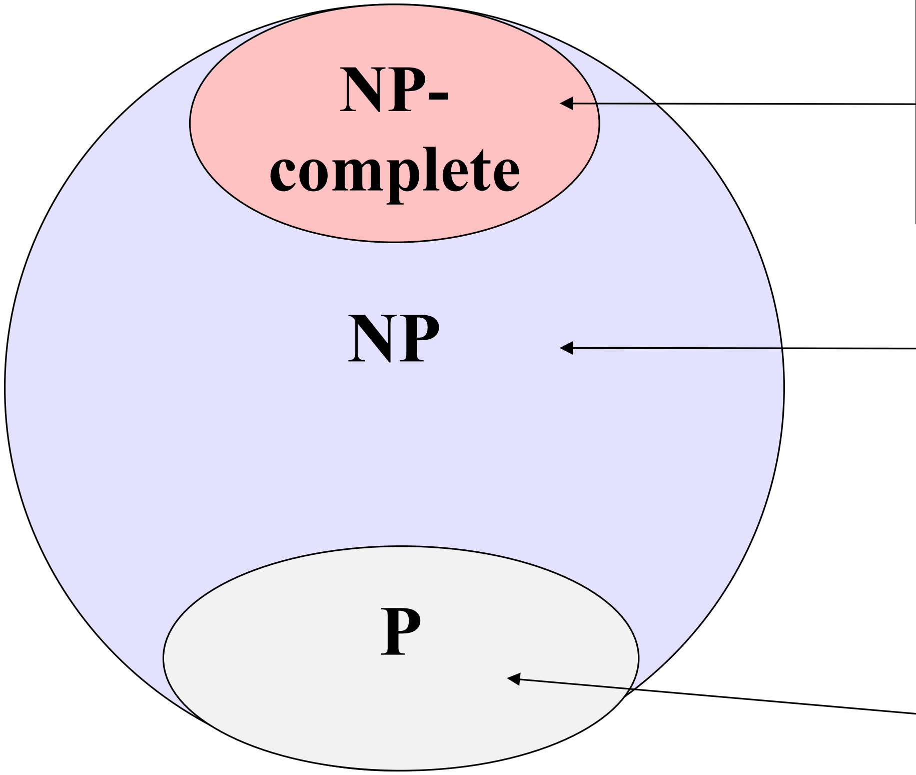
Theorem: *Any* problem in **NP** poly-time reduces to 3-SAT. Hence, 3-SAT is **NP**-hard and **NP**-complete.

The Cook-Levin Theorem

Theorem: *Any* problem in **NP** poly-time reduces to 3-SAT. Hence, 3-SAT is **NP**-hard and **NP**-complete.

Hence, so are HAM-CYCLE, SUBSET-SUM, VERTEX-COVER, INDEPENDENT-SET,…

Complexity Classes



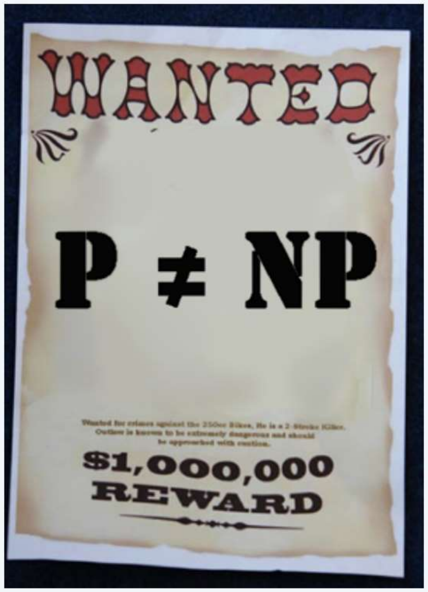
Problems that are computationally equivalent to 3-SAT

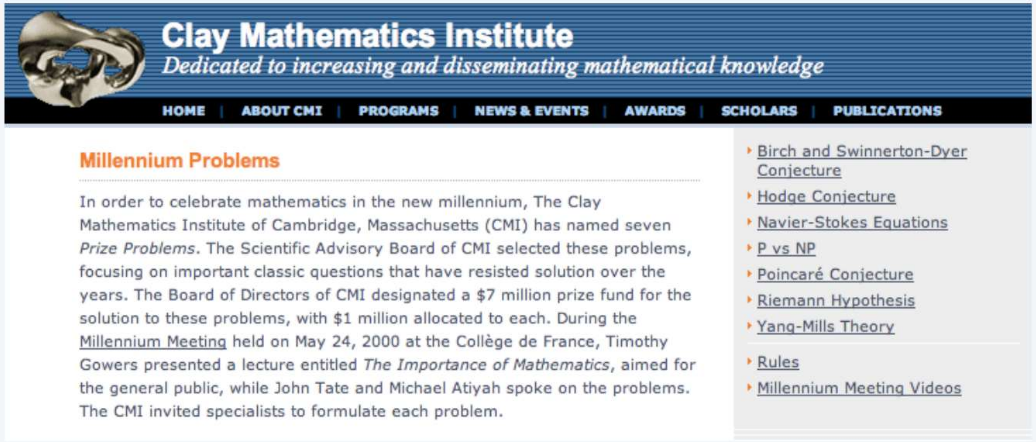
Problems that poly-time reduce to 3-SAT

Problems solvable in (deterministic) poly time

Millennium Prize

$1 million dollars for resolution of **P=NP** or **P≠NP**


WANTED
P ≠ NP
Wanted for crimes against the 250cc Bikes, He is a 2-Stroke Killer.
Outlaw is known to be extremely dangerous and should
be approached with caution.
$1,000,000
REWARD


Clay Mathematics Institute
Dedicated to increasing and disseminating mathematical knowledge
HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a $7 million prize fund for the solution to these problems, with $1 million allocated to each. During the Millennium Meeting held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

- Rules
- Millennium Meeting Videos

Some Quotes

National University of Singapore

"I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (i) It is a legitimate mathematical possibility and (ii) I do not know." — Jack Edmonds 1966

"If I had to bet now, I would bet that P is not equal to NP. I estimate the half-life of this problem at 25–50 more years, but I wouldn't bet on it being solved before 2100. " — Bob Tarjan (2002)

Some Quotes

“ I think that in this respect I am on the loony fringe of the mathematical community: I think (not too strongly!) that P=NP and this will be proved within twenty years. Some years ago, Charles Read and I worked on it quite bit, and we even had a celebratory dinner in a good restaurant before we found an absolutely fatal mistake. ” — Béla Bollobás (2002)

Extent and Impact

- Garey and Johnson’s book, “Computers and Intractability”, includes over 300 **NP-complete** problems and is the #1 cited reference in computer science!
- NP-completeness is used in more than 6,000 publications per year (more than “compiler”, “OS”, “database”)
- Main intellectual export of computer science.

Beyond NP-completeness

- Approximation algorithms: Find an approximate solution in polynomial time.
  - Simple greedy algorithm gives a 2-approximation for VERTEX-COVER.
- Use heuristics and exploit structure in typical inputs.
  - Modern SAT solvers can solve even relatively large instances quite fast.

Remainder of Semester

- Assignment 10 due on April 10
- Assignment 11 due on April 17
- Assignment 12 cancelled!
- Programming assignment 2 (optional) due April 21 (Thu)
- Final exam practice questions to be released tomorrow
  - Review session to go over these questions on April 20 (Wed), 2-4pm on Zoom

Final Exam

- 5-7pm on April 26 (Tuesday)
- Logistics similar to midterm, uploaded to LumiNUS
- Format of the exam also similar to midterm
- From material before the midterm, only material from Lectures 2 and 3 will be tested
- Suggestion: Review assignments, tutorials, and final exam practice questions

Showing NP-completeness

- There will be a question that asks you to show that a certain problem (let’s call it X) is NP-complete.
- What are the steps?
- First, show that X is in NP.
  - A YES-instance has a certificate that can be verified in polynomial time.

Showing NP-completeness

- Second, show that X is NP-hard.
- This is done by giving a polynomial-time reduction **from** another NP-hard problem A. (In the exam, I will tell you which problem A to use.)
- The reduction should convert any instance of A into an instance of X.
  - You are not trying to solve the instance of A directly.
  - The reduction should not depend on whether the instance of A is a YES-instance or a NO-instance.

Showing NP-completeness

- To show that the reduction is valid, you need to show that
  - The reduction runs in polynomial time (if this is clear, you can simply say that it is clear)
  - If the instance of A is a YES-instance, then the instance of X is also a YES-instance
  - If the instance of X is a YES-instance, then the instance of A is also a YES-instance

Showing NP-completeness

- Some tips:
  - If you are not sure how to start a reduction, try something trivial, see where it fails and how you can fix it.
  - Review reductions from lectures, assignments, tutorials.

Input Representation

- Why is the knapsack dynamic programming algorithm **not** a polynomial-time algorithm?
- The algorithm runs in time $O(nW)$.

$O(n + \lg W)$ also correct

- However, the input has size only $\underline{O(n \lg W)}$, because the number W can be represented using $\lg W$ bits.
- The running time of the algorithm is not polynomial in the size of the input.

NUS
National University of Singapore

That’s all for CS3230!

Please fill in the student feedback survey.

Good luck with the final exam (and beyond) ☺

Acknowledgements

NUS
National University of Singapore

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Lee Wee Sun
  - the slides from Prof. Kevin Wayne
  - the slides from Prof. Arnab Bhattacharyya