Introduction to NP-Completeness

CHANG Yi-Jun   Steven HALIM   PRASHANT Nalini Vasudevan

School of Computing
National University of Singapore

CS3230 Lec10; Tue, 28 Oct 2025

NUS | Computing
National University
of Singapore

Overview

NUS | Computing

Recap: Decision Problems

A **decision problem** is a function that maps an instance space I (containing instances/inputs $x \in I$) into the solution set { YES, NO }.

Input x → [Decision Problem] → YES/NO
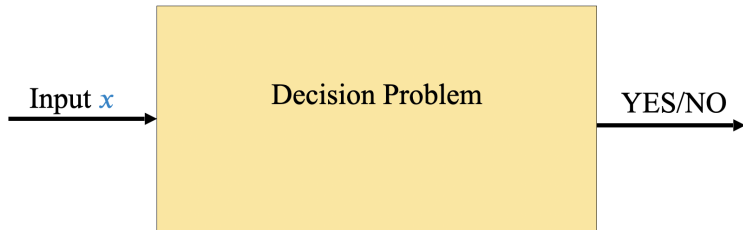
Recap: Reductions between Decision Problems

Given two decision problems A and B, a **polynomial-time Karp reduction** from A to B, denoted by $A \leq_p B$, is a transformation from instance α of A to instance β of B such that:

1. α is a YES-instance for A **iff** β is a YES-instance for B.
2. The transformation takes polynomial-time in the size of α.
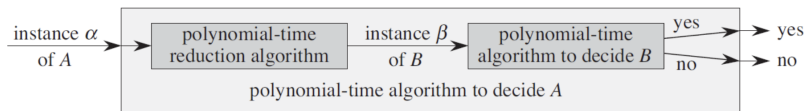
NP — A class of problems

and how it came into existence

Going back to the 1960s

Efficient algorithm was found	No efficient algorithm till date
Shortest Path (SSSP)	Longest Path
Min Spanning Tree (MST)	Traveling Salesperson Problem
Euler tour	Hamiltonian cycle
Min Cut	Balanced Cut
VC on Tree	Vertex Cover (VC)
IS on Bipartite Graph	Independent Set (IS)
Bipartite Matching	3D Matching
Linear Programming (LP)	Integer Programming (IP)

It was quite surprising and even frustrating to be unable to find efficient algorithms for so many problems when other similar-looking versions had very efficient algorithms.

Searching versus Verifying on 2 hard problems

`Longest-Path` **decision problem**

Given a graph G, does there exist a simple path of length $\geq k$?

Searching for a path of length $\geq k$ appears to be difficult.

Verifying if a given path is of length $\geq k$ is easy.

`Vertex-Cover` **decision problem**

Given a graph G, does there exist a vertex cover of size $\leq k$?

Searching for a subset of $\leq k$ vertices that is a vertex cover appears to be difficult.

Verifying if a given subset of $\leq k$ vertices is a vertex cover is easy.

Back to these hard problems

No efficient algorithm till date
Longest Path Traveling Salesperson Problem Hamiltonian cycle Balanced Cut Vertex Cover (VC) Independent Set (IS) 3D Matching Integer Programming (IP)

Searching for the solution is difficult
Verifying the solution is easy

NP class - Background

Given any decision problem X, a **verifier** for X is an algorithm A that takes as input an instance I of X (which can be a YES-instance or a NO-instance), and a certificate/proposed solution s and outputs YES/NO, <u>verifying</u> whether the certificate is is right or wrong.

Formally: I is YES-instance of X **if and only if (iff)** there exists a **string** s such that A outputs YES on input (I, s).

For verification to be efficient, A must run in **polynomial-time**.

And s must be short, $|s| \leq p(|I|)$, where p is a polynomial function.

NP class - Definition

NP: The set of all decision problems which have efficient (polynomial-time) **verifier**.

NP: "**Non-deterministic polynomial time**"

Example: Hamiltonian–Cycle (HC)

Given a graph G, does there exist a (simple) cycle that visits each vertex exactly once?

The certificate is the cycle itself.

The verifier checks whether it is a cycle (starts and ends at the same vertex) and visits each vertex once (perhaps by using a Direct Addressing Table (DAT) of `visited` flags). This runs in polynomial-time (in $O(|V|)$).

Hence, HC is in NP.

NP versus P

Recall:

NP: "**Non-deterministic polynomial time**"

NP: The set of all decision problems which have efficient (polynomial-time) **verifier**, i.e., 'easy to check'.

versus

P: "**Polynomial Time**"

P: The set of all decision problems which have an efficient (polynomial-time) **algorithm**, i.e., 'easy to solve'.

Is there any relation between P and NP?

NP class - for problems in P

Recall:

Given any decision problem X, a **verifier** for X is an algorithm A that takes as input an instance I of X (which can be a YES-instance or a NO-instance), and a certificate/proposed solution s and outputs YES/NO, <u>verifying</u> whether the certificate is is right or wrong.

If X is a problem in P,
let Q be a **polynomial-time** algorithm for **solving** X.

Thus, given input (I, s), where s is a certificate/proposed solution,
this algorithm A can ~~verify if the certificate s is right or wrong.~~
just ignore s, and execute the algorithm Q on input I.
If the answer is YES/NO, output YES/NO, respectively.

NP versus P - First Diagram

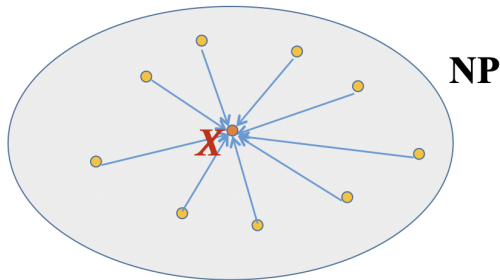NP (Verifying a proposed solution) versus P (Finding a solution)
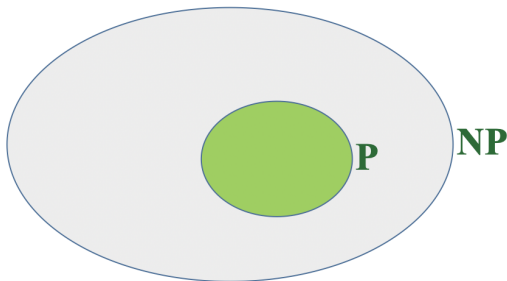
Open Question: Is **P = NP**?

NP-hard and NP-complete

NP-hard and NP-complete — two other class of problems

and how it came into existence

A problem X is in NP-hard if **for every $A \in$ NP, $A \leq_p X$.**
(X is at least as hard as any other problem in NP).

If additionally, X is also (proven to be) in NP,
then X is NP-complete.

Does any NP-complete problem exist?

It really needs **courage** to ask such a question and **great insight** to pursue its answer.

Because according to NP-complete definition: **Every problem**, known as well unknown problems, from the class NP has been reducible to this NP-complete problem.

Therefore, such an NP-complete problem would indeed be the hardest of all problems in NP.

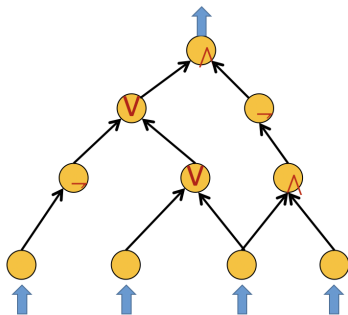Only such great questions in science lead to great inventions.

The first one: `Circuit-Satisfiability`[1] (C-SAT)

`Circuit-Satisfiability`
(`C-SAT`): [Cook and Levin, 1971]:

Given a DAG with vertices corresponding to AND, OR, NOT gates and n binary inputs, does there exist any binary input which gives output 1?

Why is `C-SAT` in NP?

The certificate is a binary input that gives output 1.

e.g., F,T,T,F for this example

For every $A \in NP$, $A \leq_p$ C-SAT

Consider any problem $A \in NP$.
What we know is that it has an efficient verifier, say Q.

Any verifier algorithm which outputs YES/NO can be represented as a C-SAT DAG where the internal vertices are the gates, the leaves are the binary inputs, and the output is 1/0 (or YES/NO).
PS: The technical details is very dry...

So Cook & Levin essentially transform Q into the corresponding DAG and simulate Q on the instance I and proposed solution s.

PS: This is just a sketch.
Interested students should study Cook-Levin theorem on your own.

Satisfiability (CNF-SAT) - Definitions

Literal:
A Boolean variable or its negation, e.g., x_i, $\overline{x_i}$.

Clause:
A disjunction (OR) of literals, e.g., $C_j = x_i \vee \overline{x_2} \vee x_3 \vee x_4$.

Conjunctive Normal Form (CNF):
A formula Φ that is a conjunction (AND) of clauses, e.g.,
$\Phi = C_1 \wedge C_2 \wedge C_3$.

CNF-SAT:
Given a CNF formula Φ, does it have a satisfying truth assignment?

3-Satisfiability (3-SAT) - Definitions

3-SAT is CNF-SAT where each clause contains exactly 3 (three) literals corresponding to different variables.

For example: $\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$

An unsatisfying assignment: $x_1 = \textit{True}$ and $x_2 = x_3 = x_4 = \textit{False}$.
PS: There are 4 other unsatisfying assignments for this Φ.

A satisfying assignment: $x_1 = x_2 = x_4 = \textit{True}$ and $x_3 = \textit{False}$.
This can act as a certificate, this 3-SAT in NP.
PS: There are 10 other satisfying assignments for this Φ.

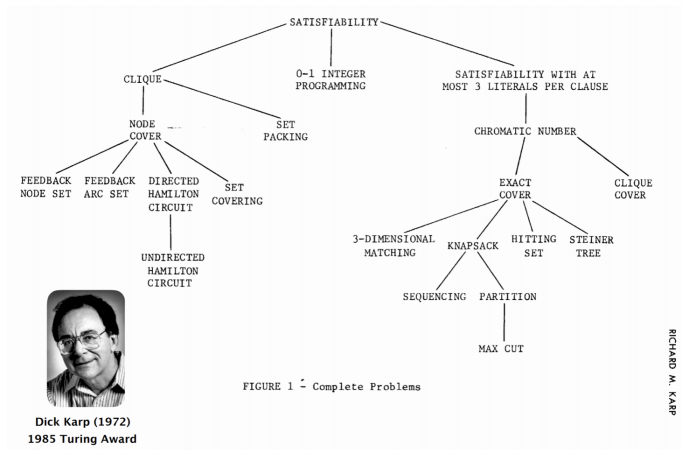PS: A short Python script that brute forces all possibilities.

3–SAT is NP-complete

Earlier, we have been shown that `C-SAT` is NP-complete.

We are not proving these, but `C-SAT` $\leq_p$ `CNF-SAT` $\leq_p$ `3-SAT`.

So, `3-SAT` is NP-complete.

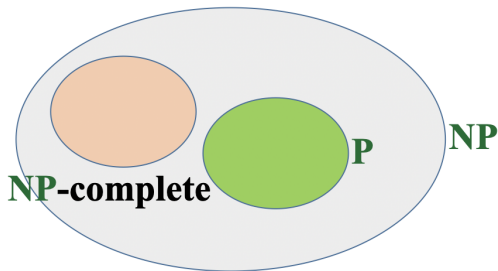# Reduction Proofs (opening the floodgates)

FIGURE 1 - Complete Problems

Dick Karp (1972)
1985 Turing Award

RICHARD M. KARP

Our version (ongoing): https://visualgo.net/en/reductions

Recall: A problem X in NP class is in NP-complete if
for every $A \in$ NP, $A \leq_p X$.

Implication: If <u>any</u> NP-complete problem is solved in
polynomial-time, then P = NP.

If you can prove P = NP (**or** P $\neq$ NP),
what will you get from Clay Mathematics Institute?

A). Turing award
B). USD $1 M
C). Nobel prize
D). Clay prize
E). Nothing

Which of the following decision problems is **not** NP-complete?

A). C-SAT

B). CNF-SAT

C). 3-SAT

D). 2-SAT

E). None of the above

Some Quotes

"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 $\neq$ NP. I estimate the half-life of this problem at 25–50 more years, but I would not bet on it being solved before 2100." — Robert Tarjan (2002)

"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)

How to show a problem to be NP-complete?

Let X be a problem which we wish to show to be NP-complete (e.g., in WA3 and/or final assessment). Follow these two steps:
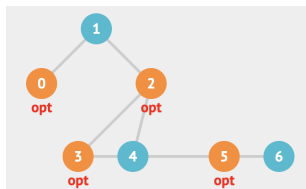
1. Show that $X \in$ NP (This is usually the easier one).
2. Pick a problem A which is already known to be NP-complete (having a library of problems that are NP-complete helps, i.e., `https://visualgo.net/en/reductions`);
   Then, show that $A \leq_p X$, i.e., X is NP-hard.
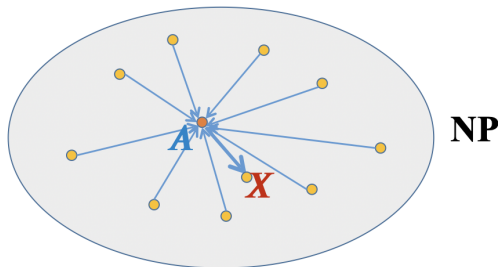
Independent-Set (IS)

Definition: Given an undirected graph $G = (V, E)$, a subset $X \subseteq V$ is said to be an **independent set** if for each $u, v \in X$, edge $(u, v) \notin E$.

Optimization version: compute `Max-Independent-Set` (MIS). See `https://visualgo.net/en/mvc` (both are very related).

For example, MVC = $\{0, 2, 3, 5\}$ and MIS = $\{1, 4, 6\}$.

Decision version: Does there exist an independent set of size $\geq k$?

Prove IS is NP-complete

1. Show that `IS` $\in$ NP
   - ▶ The certificate is the independent set.
   - ▶ We can check if the size is $\geq k$ in polynomial-time.
   - ▶ We can also check that for all edges $(u, v) \in E$, only at most u or v (but not both) is in the independent set (perhaps store the independent set information as a Hash Table). This check can be done in $O(E)$, which is also polynomial-time.

2. Pick a problem A which is already known to be NP-complete (having a library of problems that are NP-complete helps, i.e., `https://visualgo.net/en/reductions`);
   Then, show that $A \leq_p X$, i.e., X is NP-hard.
   - ▶ Any NP-complete problem that we know can be problem A.
   - ▶ But choose the one that leads to the easiest reduction.
   - ▶ We will pick `3-SAT`.

3-SAT $\leq_p$ IS

Be careful of the direction of the reduction, it is from a **known** NP-complete problem, in this case 3-SAT, to the (new) problem that you want to show to be NP-complete, in this case IS.
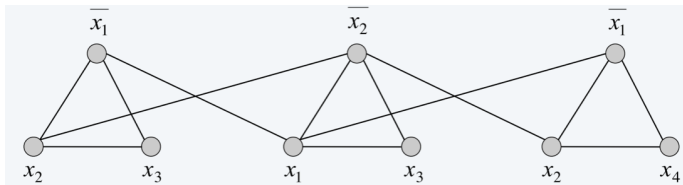
Given an instance Φ of 3-SAT, the goal is to construct an instance (G, k) of IS so that Φ is satisfiable if and only if (iff) G has an independent set of size k.

PS: It suffices to show for <u>exactly</u> k, not $\geq k$.

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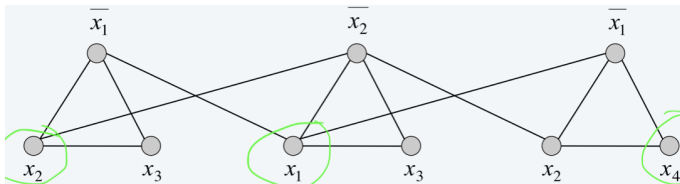
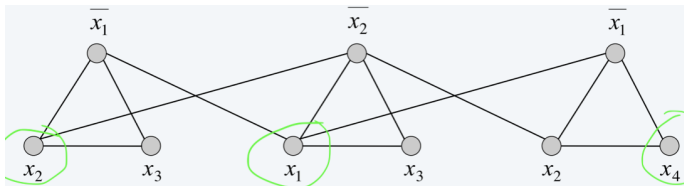$$\downarrow\uparrow$$

Do you see the required (polynomial-time) steps?

Φ is a YES (3-SAT) $\rightarrow$ (G, k) is a YES (IS)

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$

$\downarrow$

Suppose Φ is a YES-instance of 3-SAT,
then (G, k) is a YES-instance of IS.

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$

$\uparrow$

Suppose (G, k) is a YES-instance of IS,
then Φ is a YES-instance of 3-SAT.

Worst-case Analysis

The proof that we have just seen shows that <u>some instances</u> of IS are as hard to solve as the 3−SAT problem.

This does not mean that all instances of the IS problem are hard!

So, if there is no polynomial-time algorithm that solves 3−SAT on <u>all</u> instances, there is no polynomial-time algorithm that solves IS on <u>all</u> instances.

Question 3 at VisuAlgo Online Quiz

What is the best way to solve IS if the given graph G is an (unweighted) <u>Tree</u>?

A). Complete Search (Brute Force), IS is NP-complete
B). Divide and Conquer
C). Greedy Algorithm
D). Dynamic Programming
E). Randomized Algorithm
F). I do not know... something outside CS3230 syllabus?

Prove Vertex-Cover (VC) is NP-complete

Just now, we have seen that IS is NP-complete.

We also know, from the previous lecture, that IS $\leq_p$ VC.

So VC is also NP-complete
(don't forget to quickly show that VC is in NP).

Prove `Clique` is NP-complete

Presenting another problem: `Clique`: Given an undirected graph G and an integer k, does there exist a clique in G of size $\geq k$?

Exercise: Prove that `Clique` is NP-complete.
Start by proving that `Clique` is easy to verify, i.e., in NP.
Hints: Try to prove that IS $\leq_p$ `Clique`.
(Details left as homework).

Side Note: Status of 3-SAT

The fastest algorithm known ($\approx 2020s$) for 3-SAT runs in time $\approx 1.307^n$.

It is believed that there is no $2^{o(n)}$-time algorithm for 3-SAT (Exponential Time Hypothesis).

As we have seen with 3-SAT $\leq_p$ IS, it is often very convenient to reduce from 3-SAT to other problems in order to show that those other problems will also be hard if 3-SAT is hard.

See the out-degree of 3-SAT at
`https://visualgo.net/en/reductions`.

Hitting-Set (HS)

Read https://visualgo.net/en/reductions?slide=19.

Given a set of sets $S = \{S_1, S_2, \ldots, S_n\}$, a set H is said to be a
<u>hitting set</u> of S if for all S_i, $H \cap S_i \neq \emptyset$ (i.e., H has a non-empty
intersection with <u>all</u> S_i).

Given the set of sets S and a non-negative integer k,
the Hitting-Set (HS) problem asks:
Is there exists a hitting set of S of size $\leq k$?

e.g., $S = \{S_1, S_2, S_3, S_4\}$,
$S_1 = \{2, 3\}$, $S_2 = \{6, 7\}$, $S_3 = \{1, 4, 5, 7\}$, $S_4 = \{3, 7, 8\}$, $k = 2$,
then it is a YES-instance with certificate $H = \{3, 7\}$.

Show that the Hitting-Set Problem is NP-complete.
Hints: Try a reduction from Vertex-Cover.

Hitting-Set is in NP

A hitting set of size at most k can act as a certificate.

The size of this certificate is polynomial.

It is easy to verify this certificate in polynomial-time
(by checking its size is $\leq k$
and whether it has a non-empty intersection with all S_i).

VC $\leq_p$ HS

Read https://visualgo.net/en/reductions?slide=31 and
https://visualgo.net/en/reductions?slide=31-1.

Take an instance (G, k) of vertex cover.
For each edge $e = (u, v)$ in G, create a set $S_e = \{u, v\}$.
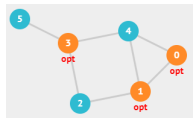The instance of hitting set is: $\{S_e | e$ is an edge in $G\}$, k.
For example, given the following graph G with VC $=$
$\{0, 1, 3\}, k = 3$, click here to open VisuAlgo MVC page.
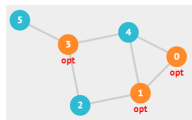
Create $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$, $S_1 = \{5, 3\}$, $S_2 = \{3, 4\}$,
$S_3 = \{3, 2\}$, $S_4 = \{2, 1\}$, $S_5 = \{4, 1\}$, $S_6 = \{4, 0\}$, $S_7 = \{1, 0\}$,
$k = 3$, so $H = \{0, 1, 3\}$.

$\downarrow\uparrow$

Create $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$, $S_1 = \{5, 3\}$, $S_2 = \{3, 4\}$, $S_3 = \{3, 2\}$, $S_4 = \{2, 1\}$, $S_5 = \{4, 1\}$, $S_6 = \{4, 0\}$, $S_7 = \{1, 0\}$, $k = 3$.

Clearly, the transformation takes polynomial-time, in $O(E)$.

It is easy to see that G has a vertex cover of size at most k **if and only if** $\{S_e | e$ is an edge in $G\}$ has a hitting set of size at most k, simply because the elements in the hitting set correspond to the vertices in the vertex cover.

Impact?

Garey and Johnson's book, "Computers and Intractability", includes over 300 NP-complete problems and is the number 1 cited reference in computer science!

NP-completeness is used in more than 6000 publications per year (more than 'compiler', 'OS', 'database').

Main intellectual export of Computer Science.

Computational Complexity Course(s)

There are problems that are provably harder than NP-complete problems, problems that require polynomial space, problems that require large circuits, problems that are unsolvable even with unlimited time!

Enter the world of complexity theory. . .

Take Computational Complexity courses (CS3231→CS5230)!

Recap of this lecture (1)
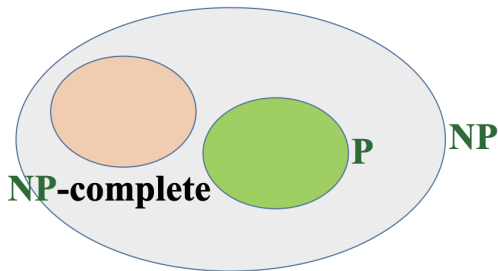
How to show a problem to be NP-complete?

Let X be a problem which we wish to show to be NP-complete

1. Show that $X \in$ NP.
2. Pick a problem A which is already known to be NP-complete; (having a library of problems that are NP-complete helps, i.e., `https://visualgo.net/en/reductions`); Then, show that $A \leq_p X$, i.e., X is NP-hard.

NP
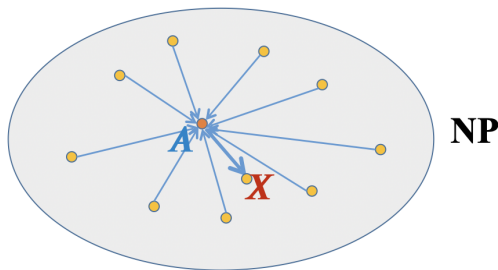
Recall: A problem X in NP class is in NP-complete if
for every $A \in$ NP, $A \leq_p X$.

Implication: If <u>any</u> NP-complete problem is solved in
polynomial-time, then P = NP.

Problems in NP but believed not to be NP-complete

There are some problems in NP but believed not to be NP-complete, e.g., `Integer-Factoring`:
Given an integer, compute all its prime factors.
Decision version: Given an integer n and two integers $2 \le d_1 < d_2 < n$, is there any prime factor of n in $[d_1, d_2]$?

It belongs to NP: Given any integer $x \in [d_1..d_2]$, it is possible, in polynomial-time, to determine if x is a prime and to determine if x divides n.

`Integer-Factoring` is believed to be <u>more difficult</u> than the problems in P, and <u>easier</u> than problems that are NP-complete. There is no proof of this till now and the research on this continues.

Optimisation Algorithms Course (CS4234)

What to do when a problem is NP-complete?

Unless P=NP,
NP-complete problems have no poly-time algorithms.

But they come up frequently in real-life, so what can we do?

- ▶ We can try to solve <u>smaller</u> instances optimally using (still) exponential time algorithms (brute force or cleverer methods such as branch and bound), e.g., if it is a 0/1-Knapsack problem with a <u>small capacity</u> W, then we can use the pseudo-polynomial DP algorithm.
- ▶ We can check if the problem instance has special features that make it more efficiently solvable, e.g., 2-SAT solvers.
- ▶ We can design a (greedy) <u>approximation</u> or <u>local search</u> algorithm.

Let's solve /house-robber-iii

Acknowledgement

The slides are modified from previous editions of this course and similar course elsewhere

NUS | Computing