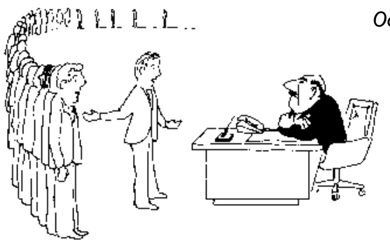
15-251
Great Ideas in Theoretical Computer Science

Lecture 18:
NP and NP-completeness continued

October 26th, 2017



I can't find an efficient algorithm, but neither can all these famous people.

A Quick Review

Exponential running time examples

Theorem Proving Problem

(informal description)

Given a mathematical proposition P and an integer k, determine if P has a proof of length at most k.

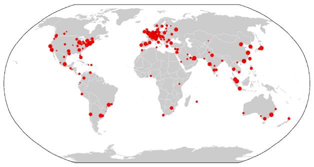
Subset Sum Problem

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exponential running time examples

Traveling Salesperson Problem (TSP)



Is there an order in which you can visit the cities so that ticket cost is < $50000?

Exponential running time examples

Satisfiability Problem (SAT)

Input: A Boolean propositional formula.
e.g. $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3 \wedge x_4) \vee x_3$

Output: Yes iff there is an assignment to the variables that makes the formula True.

Circuit Satisfiability Problem (Circuit-SAT)

Input: A Boolean circuit.

Output: Yes iff there is an assignment to the input gates that makes the circuit output 1.

Some other examples

Longest Common Subsequence

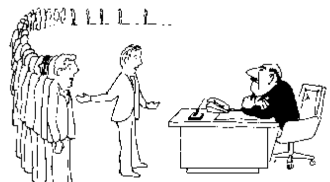
Input: A set of sequences, and a number k.
Output: Yes iff there is a subsequence of length at least k that is common to all the given sequences.

Longest Path

Input: A graph and an integer k.
Output: Yes iff there a path in G of length at least k.

So you come across one of these problems,
what do you do?

Could they be in **P**?



I can't find an efficient algorithm, but neither can all these famous people.

Is there a deep reason why these problems
all seem to be hard?

Define a complexity class

What would be a reasonable definition for:

"class of problems decidable using Brute-Force Search" ?

What is common about
SAT, Theorem Proving, TSP, Longest Path, etc…?

The complexity class **NP**

Super Informal:

NP is a set of languages that we come across all the time and would love to solve in poynomial time.

Semi-Informal:

A language is in **NP** if:

whenever we have a Yes input/instance,
there is a "*simple*" proof (solution) for this fact.

↓

1. The length of the proof is polynomial in the input size.
2. The proof can be verified/checked in polynomial time.

The complexity class **NP**

Formally:

Definition:

A language A is in **NP** if

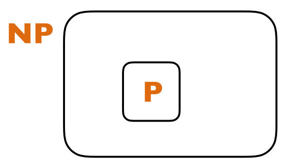
- there is a polynomial-time TM V
- a constant k

such that for all $x \in \Sigma^*$:

$$x \in A \iff \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x,u) = 1.$$

If $x \in A$, there is some poly-length proof that leads V to accept.

If $x \notin A$, every "proof" leads V to reject.



But we still don't know if *SAT*, *TSP*, *Theorem Proving*, …
are in **P** or not.

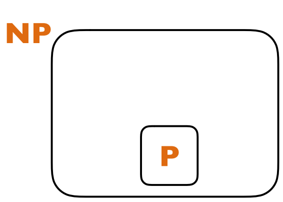
Even if we believe **P** ≠ **NP** these problems could still be in **P**.

Definitions of **NP**-hard and **NP**-complete

Definition (hardness):

We say that language A is **NP**-hard if
for all $L \in$ **NP**, $L \leq_T^P A$.

"A is at least as hard as every language in **NP**."



$\leq_T^P A$

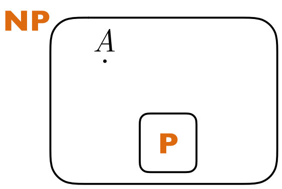
Definitions of **NP**-hard and **NP**-complete

Definition (completeness):

We say that language A is **NP**-complete if

- A is **NP**-hard;
- $A \in$ **NP**.

"A is a representative for hardest languages in **NP**."



$\leq_T^P A$

Definitions of **NP**-hard and **NP**-complete

Observation:

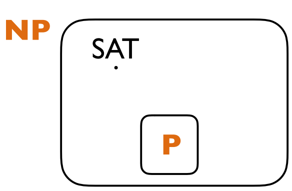
Suppose A is **NP**-complete.

- If **NP** = **P**, then $A \in$ **P**.
- If $A \in$ **P**, then **NP** = **P**.

NP = **P** $\iff A \in$ **P**

2 possible worlds

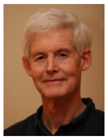
Could it be true that one of
SAT, Theorem Proving, TSP, Sudoku, etc.
is **NP**-complete?



$\overset{?}{\leq_T^P}$ SAT

Is there **any** language that is **NP**-complete??

The Cook-Levin Theorem



Theorem (Cook 1971 - Levin 1973):

CIRCUIT-SAT is **NP**-complete.

So CIRCUIT-SAT is in **NP**. (easy)

And for every L in **NP**, L $\leq_T^P$ CIRCUIT-SAT.

Karp's 21 **NP**-complete problems

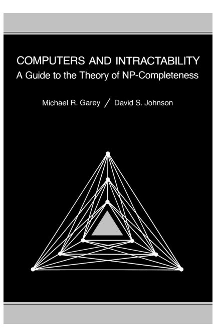
1972: "Reducibility Among Combinatorial Problems"

0-1 Integer Programming	Partition
Clique	Clique Cover
Set Packing	Exact Cover
Vertex Cover	Hitting Set
Set Covering	Knapsack
Feedback Node Set	Steiner Tree
Feedback Arc Set	3-Dimensional Matching
Directed Hamiltonian Cycle	Job Sequencing
Undirected Hamiltonian Cycle	Max Cut
3SAT	Chromatic Number

Today

Thousands of problems are known to be **NP**-complete.

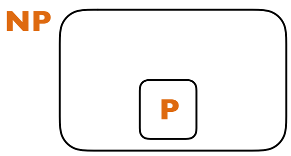
(including the problems mentioned at the beginning of lecture)



1979

How do you show a language is **NP**-complete?

How did Cook and Levin do it ?!?

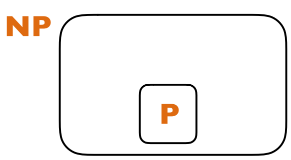


$\le_T^P$ CIRCUIT-SAT

How did Karp do it ?!?

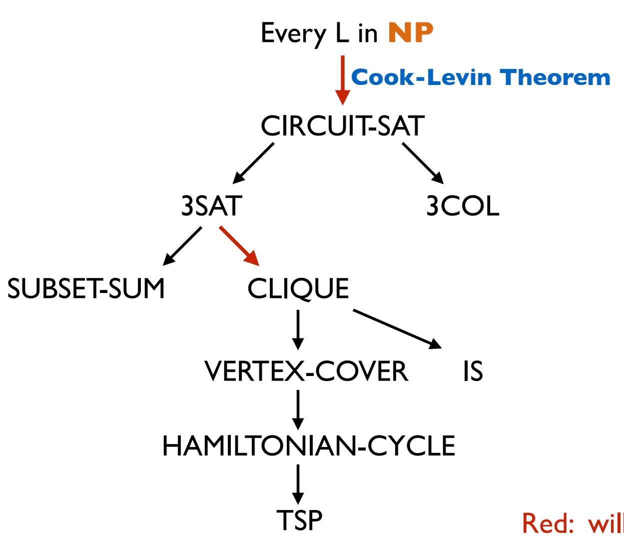
IMPORTANT NOTE:

How do you show a language is **NP**-complete?



$\le_T^P$ CIRCUIT-SAT

To show L is **NP**-hard:



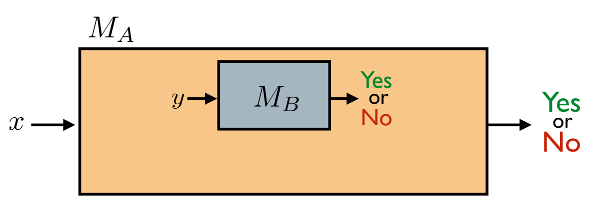
Red: will show

First:
An important note about reductions

Cook reduction

Cook reductions: poly-time Turing reductions

$$A \leq_T^P B$$



"You can solve A in poly-time using a blackbox that solves B."

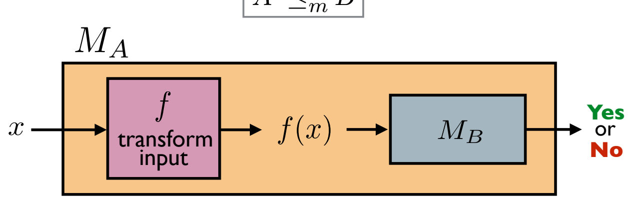
You can call the blackbox poly(|x|) times.

Karp reduction

NP-hardness is usually defined using Karp reductions.

Karp reduction (polynomial-time many-one reduction):

$$A \leq_m^P B$$



Make one call to M_B and directly use its answer as output.

We must have:

Karp reduction

Definition:

Karp reduction

Can define **NP**-hardness with respect to $\leq_T^P$.
(what some courses use for simplicity)

Can define **NP**-hardness with respect to $\leq_m^P$.
(what experts use)

These lead to different notions of **NP**-hardness.

Poll 1

Which of the following are true?

- if $A \leq_m^P B$ and $B \leq_m^P C$, then $A \leq_m^P C$.
- $A \leq_m^P B$ if and only if $B \leq_m^P A$.
- if $A \leq_m^P B$ and $B \in$ **NP**, then $A \in$ **NP**.

CLIQUE is NP-complete

Want to show:

- CLIQUE is in **NP**.
- CLIQUE is **NP**-hard.

  3SAT is **NP**-hard, so show 3SAT $\leq_m^P$ CLIQUE.

CLIQUE is in **NP**

CLIQUE

Input: $\langle G, c \rangle$ where G is a graph and c is a positive int.

Output: Yes iff G contains a clique of size c.

Fact: CLIQUE is in **NP**.

CLIQUE is in **NP**

Proof: We need to show a verifier TM V exists as specified in the definition of **NP**.

def $V(x, u)$:

CLIQUE is in **NP**

Proof (continued):

Need to show:

Definition of 3SAT Problem

3SAT

Input: A Boolean formula in "conjunctive normal form" in which every clause has exactly 3 literals.

e.g.:

$$\underbrace{(x_1 \vee \neg x_2 \vee x_3)} \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

a clause (an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

(Note: To satisfy the formula, you need to satisfy each clause.)

Output: Yes iff the formula is satisfiable.

Aside: 3SAT is in **NP**

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

φ satisfiable

$\Longleftrightarrow$

can pick one literal from each clause and set them to True

$\Longleftrightarrow$

the sequence of literals picked does not contain both a variable and its negation.

What is a good proof that $\varphi \in$ 3SAT ?

- a truth assignment to the variables that satisfies the formula.
- a sequence of literals, one from each clause, that does not contain both a variable and its negation.

CLIQUE is **NP**-complete: High level steps

CLIQUE is in **NP**. ✓

We know 3SAT is **NP**-hard.
So suffices to show 3SAT $\leq_m^P$ CLIQUE.

We need to:

3SAT $\leq$ CLIQUE: Defining the map

1. Define a map $f : \Sigma^* \to \Sigma^*$.

not valid encoding of a 3SAT formula $\mapsto$ ϵ

otherwise we have valid 3SAT formula φ (with *m* clauses).

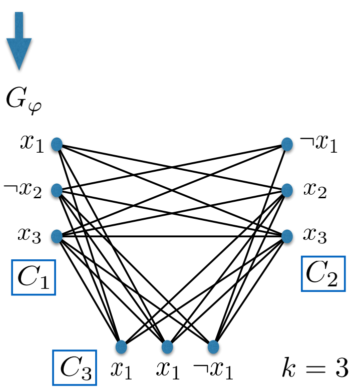
$\varphi \mapsto \langle G, k \rangle$ (we set $k = m$)

Construction demonstrated with an example.

3SAT $\leq$ CLIQUE: Defining the map

C_1 $\wedge$ C_2 $\wedge$ C_3

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_1 \vee \neg x_1)$$



The construction:

- A vertex for each literal in each clause.
- No edges between two literals in the same clause.
- No edges between x_i and $\neg x_i$ for any i.
- All other possible edges present.
- Set k to be # clauses in φ.

3SAT $\leq$ CLIQUE: Why it works

If φ is satisfiable, then G_φ contains an m-clique:

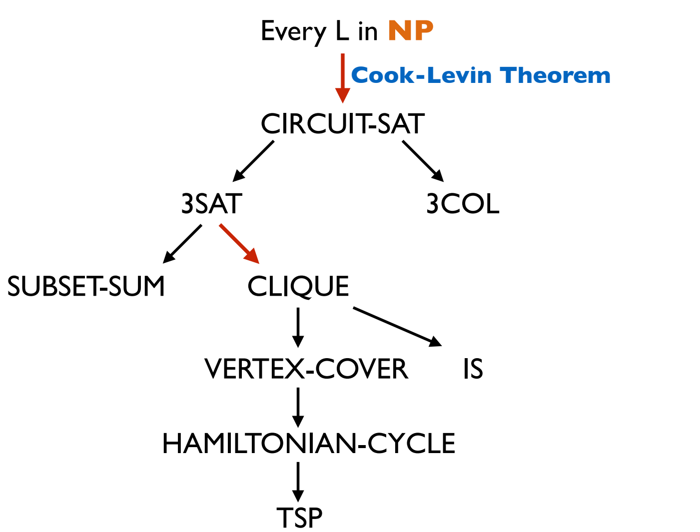
3SAT ≤ CLIQUE: Why it works

If G_φ contains an m-clique, then φ is satisfiable:

3SAT ≤ CLIQUE: Poly-time reduction?

Creation of G_φ is poly-time:

□



CIRCUIT-SAT is NP-complete

Recall

Theorem: Let $f : \{0,1\}^* \to \{0,1\}$ be a decision problem which can be decided in time $O(T(n))$.
Then it can be computed by a circuit family of size $O(T(n)^2)$.

With this Theorem, it is actually easy to prove that

CIRCUIT-SAT is **NP**-hard.

Proof Sketch