15-251: Great Ideas in Theoretical Computer Science

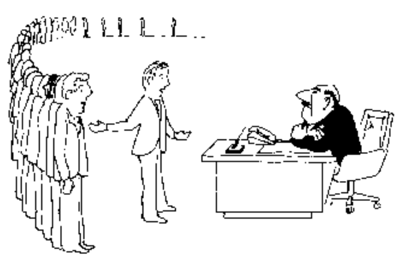
Lecture 16: NP and NP-completeness 2

October 23rd, 2018

A Quick Review

You come across a problem you cannot solve.
What do you do?

Could it be in **P**?

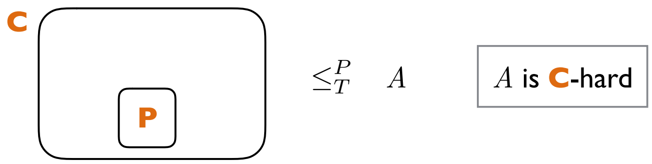


I can't find an efficient algorithm, but neither can all these famous people.

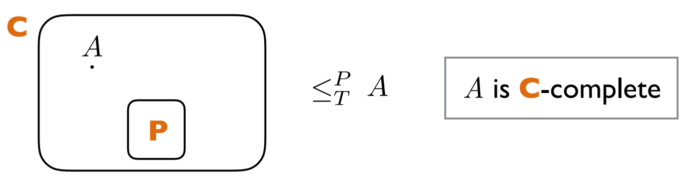
Is there a deep reason why the problem seems to be hard?

Summary so far

- How do you identify *intractable* problems?
  (problems not in **P**) e.g. *SAT, TSP, Subset-Sum*, …
- Poly-time reductions $A \leq_T^P B$ are useful to compare hardness of problems.
- Evidence for intractability of A:
  Show $L \leq_T^P A$, for **all** $L \in$ **C**, for a large class **C**.



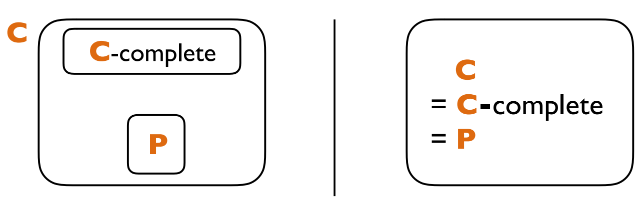
Summary so far



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

Summary so far

2 possible worlds



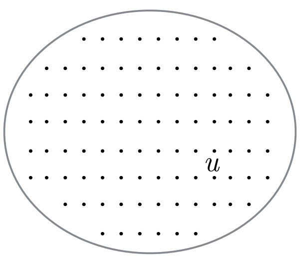
Summary so far

- The complexity class **NP** (take **C** = **NP**)

Summary so far

Which languages L are in **NP**?

1. Every x in L has
   (at most) exponentially large "*possible solutions space*"

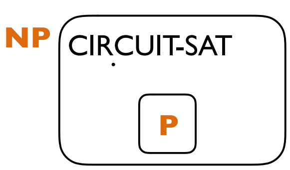


2. Easy (poly-time) to verify whether a possible solution is indeed a solution or not.

Summary so far

- **NP**-hardness, **NP**-completeness

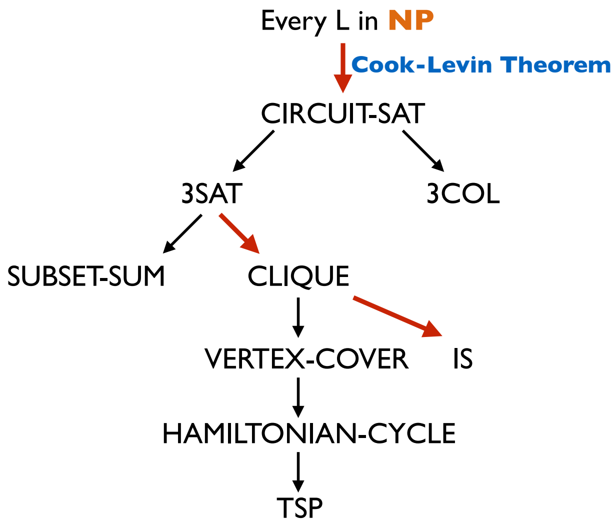
- Cook-Levin Theorem: CIRCUIT-SAT is **NP**-complete



$\leq_T^P$ CIRCUIT-SAT

- Many other languages are **NP**-complete.

- The **P** vs **NP** question

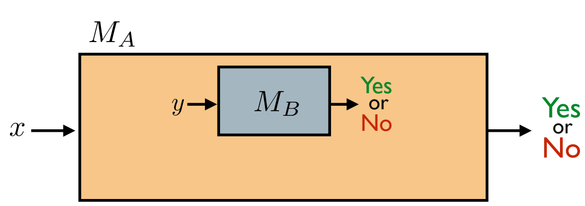


First:
An important note about reductions

Cook reduction

Cook reductions: poly-time Turing reductions

$$\boxed{A \ \le_T^P \ B}$$



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

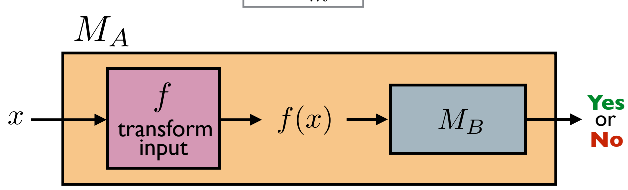
You can call the blackbox poly(|x|) times.

Karp reduction

NP-hardness is usually defined using Karp reductions.

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

$$\boxed{A \leq_m^P B}$$



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

<u>We must have:</u>

Karp reduction picture

Karp reduction: Example

CLIQUE

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

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

INDEPENDENT-SET (IS)

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

Output: Yes iff G contains an independent set of size k.

Fact: CLIQUE $\leq_m^P$ IS.

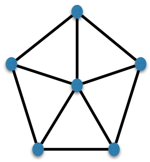
Karp reduction: Example

Want:

$$\langle G, k \rangle \overset{f}{\mapsto} \langle G', k' \rangle$$

G has a clique of size k **iff** G' has an ind. set of size k'

G



G'

Karp reduction: Example

Proof:

We need to:

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ CLIQUE $\implies$ $f(w) \in$ IS
3. Show $w \notin$ CLIQUE $\implies$ $f(w) \notin$ IS
   (often easier to argue the contrapositive)
4. Argue f is computable in polynomial time.

Karp reduction: Example

Proof (continued):

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

def $f(w)$**:**

Karp reduction: Example

Proof (continued):

2. Show $w \in$ CLIQUE $\implies f(w) \in$ IS

Karp reduction: Example

Proof (continued):

3. Show $w \notin$ CLIQUE $\implies f(w) \notin$ IS

(Show the contrapositive.)

Karp reduction: Example

Proof (continued):

4. Argue f is computable in polynomial time.

- checking if the input is a valid encoding can be done in polynomial time.
  (for any reasonable encoding scheme)

- creating E^*, and therefore G^*, can be done in polynomial time.

$\square$

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

Which of the following are true?

- 3COL $\leq_m^P$ 2COL is known to be true.
- 3COL $\leq_m^P$ 2COL is known to be false.
- 3COL $\leq_m^P$ 2COL is open.
- 2COL $\leq_m^P$ 3COL is known to be true.
- 2COL $\leq_m^P$ 3COL is known to be false.
- 2COL $\leq_m^P$ 3COL is open.
- 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.

Definition of 3SAT

3SAT

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

e.g.:

$$\underbrace{(x_1 \vee \neg x_2 \vee x_3)}_{\text{a clause (an OR of literals)}} \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

a clause
(an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

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 ?

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:

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ 3SAT $\implies$ $f(w) \in$ CLIQUE
3. Show $w \notin$ 3SAT $\implies$ $f(w) \notin$ CLIQUE
4. Argue f is computable in polynomial time.

3SAT ≤ 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 $k = m$)

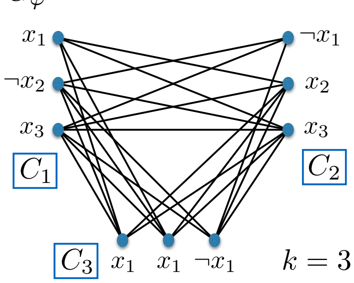
Construction demonstrated with an example.

3SAT ≤ CLIQUE: Defining the map

$$\boxed{C_1} \quad \wedge \quad \boxed{C_2} \quad \wedge \quad \boxed{C_3}$$

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

G_φ



$k = 3$

3SAT $\leq$ CLIQUE: Why it works

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

φ is satisfiable $\implies$

$\implies$ G_φ contains an m-clique.

3SAT $\leq$ CLIQUE: Why it works

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

G_φ has a clique K of size m $\implies$

$\implies$ φ is satisfiable.

3SAT $\leq$ CLIQUE: Poly-time reduction?

Creation of G_φ is poly-time:

Creating the vertex set:

- there is just one vertex for each literal in each clause.
- scan input formula and create the vertex set.

Creating the edge set:

- there are at most O(m^2) possible edges.
- scan input formula to determine if an edge should be present.

$\square$

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 $O(T(n)^2)$.

Given a TM V, we can create a circuit family that has the same behavior as V.

With this Theorem, it is actually easy to prove that CIRCUIT-SAT is **NP**-hard.

Proof Sketch

WTS: for an arbitrary L in **NP**, L $\leq_m^P$ CIRCUIT-SAT.