CSE 135: Introduction to Theory of Computation NP-completeness

Sungjin Im

University of California, Merced

04-15-2014

Significance of the question if $P \neq_? NP$

Perhaps you have heard of (some of) the following terms: NP, NPC, P, NP-hard, co-NP, EXP.

And the question if $P = NP$ or not.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.
- Problems in different complexity classes may have different challenges.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.
- Problems in different complexity classes may have different challenges.
- NP, NPC problems are widely found in practice.

Significance of the question if $P \neq_? NP$

The Millennium Prize Problems are seven problems in mathematics that were stated by the Clay Mathematics Institute in 2000. As of July 2013, six of the problems remain unsolved. A correct solution to any of the problems results in a US $1,000,000 prize (sometimes called a Millennium Prize) being awarded by the institute. The Poincare conjecture was solved by Grigori Perelman, but he declined the award in 2010.

The question $P \neq_? NP$ is the first in the list...

(source: wikipedia)

Class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$P = \bigcup_k Time(n^k)$$

Motivation: To define a class of problems that can be solved efficiently.

- P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing Machine.
- P roughly corresponds to the class of problems that are realistically solvable on a computer.

Class P

Justification

P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing Machine.

Example.

Theorem

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

In fact, it can be shown that all reasonable determinitic computational models are polynomially equivalent.

Class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$P = \bigcup_k Time(n^k)$$

Hence, a language is in P if and only if one can write a pseudo-code that decides the language. (The code must terminate for any input).

Class P

Examples

- $PATH = \{< G, s, t > \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$.
- Every context-free language.

Class *NP*

Consider the following problem of testing whether a directed graph contains a Hamiltonian path connecting two given nodes. A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.

$HAMPATH = \{< G, s, t > \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$.

We are not aware of any algorithm that decides *HAMPATH* in polynomial time. But we know a brute-force algorithm finds a s-t Hamiltonian path in exponential time. Also we know that the solution can be easily verified.

Class *NP*

Verifier

Although we do not know an instance (string) w is in *HAMPATH* or not, if it is the case, then there is a proof – here obviously, a s-t Hamiltonian path. Can we check it efficiently (in polynomial time)? Absolutely!

However, not all problems are polynomially verifiable. Think about $\overline{HAMPATH}$. How can we be convinced that a given graph has no s-t Hamiltonian path?

Class *NP*

Polynomial time Verifier

Definition

A verifier for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } < w, c > \text{ for some string } c\}$$

(c is called a certificate or proof).
We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w. A language A is polynomially verifiable if it has a polynomial time verifier.

Note that c has polynomial length in $|w|$.

Class *NP*

Polynomial time Verifier

Definition

NP is the class of languages that have polynomial time verifiers.

Class *NP*

Polynomial time Verifier

Class *NP*

Polynomial time Verifier

The term *NP* comes from nondeterminitic polynomial time and is derived an alternative characterization by using nondeterministic polynomial time Turing machines.

Theorem

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Proof.

($\Rightarrow$) Convert a polynomial time verifier V to an equivalent polynomial time NTM N. On input w of length n:

- Nondeterministically select string c of length at most n^k (assuming that V runs in time n^k).
- Run V on input $< w, c >$.
- If V accepts, accept; otherwise, reject.

Class *NP*

Theorem

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Proof.

($\Leftarrow$) Convert a polynomial time NTM N to an equivalent polynomial time verifier V. On input w of length n:

- Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
- If this branch of N's computation accepts, accept; otherwise, reject.

Class *NP*

Examples

A clique is an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.

$$CLIQUE = \{< G, k > \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Lemma

CLIQUE is in NP.

Proof.

Let $w =< G, k >$. The certificate c is a k-clique. Can easily test if c is a clique in polynomial time in w and c, and has k nodes. $\square$

Class *NP*

Examples

$HAMPATH = \{< G, s, t > \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}.$

Lemma

HAMPATH is in NP.

P versus NP?

Either of the following two must be true:
$P = NP$ or $P \subsetneq NP$.

We know that

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$$

But we do not even know if $NP \subseteq EXPTIME$ or $NP \subsetneq EXPTIME$

NP-completeness

Cook-Levin Theorem

Most researchers, however, belive that $P \neq NP$ becuase of the existence of some problems that capture the entire NP class.

A Boolean formula is an expression involving Boolean variables and operations. For example, $\Phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$ is a boolean formula. We say a boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. Note that Φ is satisfialbe $(x = 0,\ y = 1,\ z = 0)$.

Satisfiability problem (SAT)
$SAT = \{< \Phi > \mid \Phi \text{ is a satisfiable Boolean formula}\}$

Theorem (Cook-Levin Theorem)

$SAT \in P$ *iff* $P = NP$.

NP-completeness

Polynomial Time Reducibility

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

For simplicity, you can use RAM instead of Turing machine here.

NP-completeness

Polynomial Time Reducibility

Definition

Language A is polynomial time mapping reducible, or simply polynomial time reducible, to language B, written $A \leq_p B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

The function is called the polynomial time reduction of A to B.

NP-completeness

Polynomial Time Reducibility

Theorem

If $A \leq_P B$ and $B \in P$, then $A \in P$.

NP-completeness

Polynomial Time Reducibility

Theorem

If $A \leq_P B$ and $B \in P$, then $A \in P$.

Proof.

$B \in P \rightarrow$ there is a poly-time algorithm M that decides B.
$A \leq_P B \rightarrow$ there is a polynomial time reduction f from A to B.
We give an algorithm N that decides A: On input w,

- Compute $f(w)$.
- Run M on input $f(w)$ and output whatever M outputs.

We need to show two:

- Correctness: $w \in A$ iff $f(w) \in B$ iff M accepts $f(w)$. Also note that N always terminates.
- N runs in polynomial time: computing $f(w)$ takes polynomial time in $|w|$. So $|f(w)|$ is polynomial in $|w| = n$, and M is a poly-time algorithm.

NP-completeness

Definition

Definition

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. for **every** $A \in NP$, $A \leq_P B$.

NP-completeness

Definition

Definition

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. for **every** $A \in NP$, $A \leq_P B$.

Theorem

If B is NPC (NP-complete) and $B \in P$, then $P = NP$.

NP-completeness

Definition

Let's say that $B \in NPC$ if B is NP-complete.

Theorem

A language C is NP-complete if $B \leq_P C$ for some $B \in NPC$, and $C \in NP$.

NP-completeness

Cook-Levin Theorem

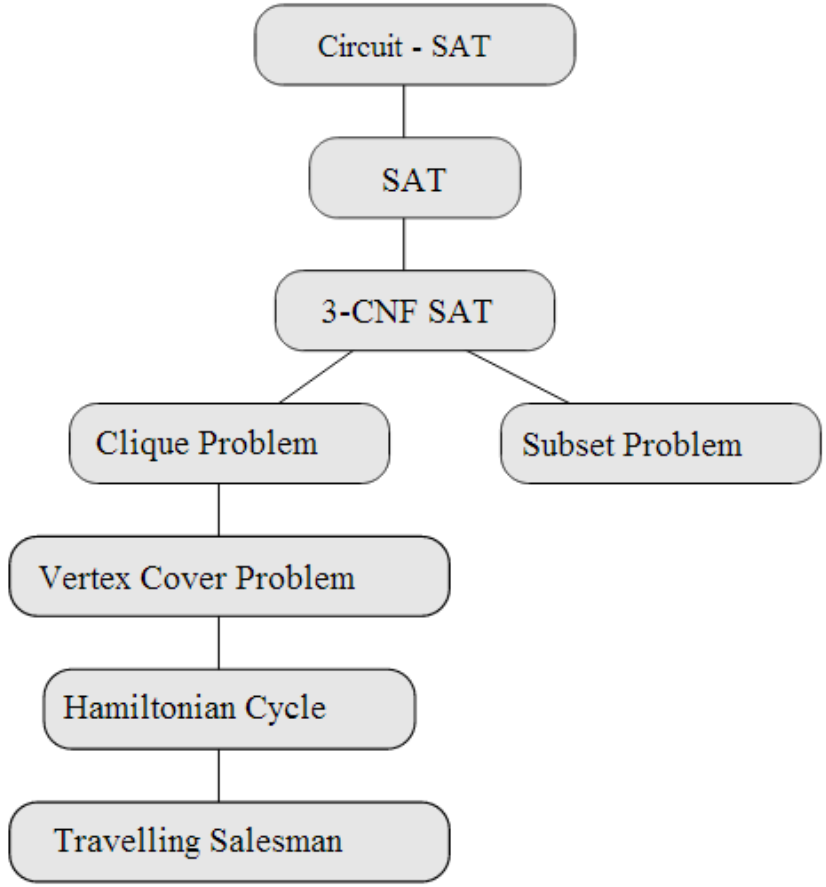
Theorem

SAT is NP-complete.

NP-completeness

NPC derivation tree

Theorem
SAT is NP-complete.



NP-completeness

NPC derivation tree

Suppose that we know $3SAT$ is NPC. Then we can show that $CLIQUE$ is also NPC.

A literal is a Boolean variable or a negated Boolean variable, as in x or $\bar{x}$. A clause is several literals connected with $\wedge$s, as in $x_1 \wedge \bar{x_2} \wedge x_4$. A Boolean formula is in conjunctive normal form, or cnf-formula, if it consists of several clauses connected with $\vee$s. It is a 3cnf-formaula if all the clauses have three literals, as in

$$(x_1 \vee x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2)$$

NP-completeness

NPC derivation tree

Theorem

CLIQUE is NPC.