Complexity Theory II

Chanathip Namprempre

Faculty of Engineering
Thammasat University

Outline

1. Example decision problems
2. Complexity class NP
   - Examples
3. P = NP???
   - NP-Completeness
   - Polynomial time reducibility
   - Proving a problem NP-complete

There are many problems that appear to be “difficult” to solve, i.e. it seems to be hard to find solutions that take polynomial time to solve them.

Moreover, some of these hard problems have related counterparts that are easy to solve.

Easy (solvable in polynomial time)	Hard
shortest path between 2 nodes in a graph	longest path between 2 nodes in a graph
traversing a graph using each edge exactly once	traversing a graph using each vertex exactly once
figuring out if a 2-CNF formula is satisfiable	figuring out if a 3-CNF formula is satisfiable

But some of the hard problems can be easily verified, i.e. verifiable in polynomial time.

Example

1. HAMPATH
2. COMPOSITES

In contrast, some hard problems do not seem to be easily verifiable, e.g. $\overline{\text{HAMPATH}}$. What would be a certificate???

Outline

Many decision problems

We have seen PATH, RELPRIME, HAMPATH, COMPOSITES

There are many more decision problems.

Example

$$
\begin{aligned}
\text{EMPTYREG} &= \{E \mid E \text{ is a regular expression and } L(E) = \emptyset\} \\
\text{NEREG} &= \{E \mid E \text{ is a regular expression and } L(E) \neq \Sigma^*\} \\
\text{CLIQUE} &= \{\langle G, k\rangle \mid G \text{ is a undirected graph with a } k\text{-clique }\} \\
\text{IS} &= \{\langle G, k\rangle \mid G \text{ has an independent set of size } k\} \\
\text{VC} &= \{\langle G, k\rangle \mid G \text{ has a vertex cover of size } k\} \\
\text{HC} &= \{\langle G\rangle \mid G \text{ has a hamiltonian cycle }\} \\
\text{TSP} &= \{\langle G, C, b\rangle \mid G \text{ is a graph, } C \text{ is a cost metrix,} \\
&\quad\ b \text{ is a non-negative integer such that} \\
&\quad\ G \text{ has a hamiltonian cycle of cost at most } b\} \\
\text{SAT} &= \{\langle \phi\rangle \mid \phi \text{ is a satisfiable boolean formula }\}
\end{aligned}
$$

Boolean logic

- A formula is an expression made up of boolean variables (e.g. x_1, x_2, y, z) connected via boolean operators (e.g. $\vee, \wedge, \neg$).

Example (Formula)

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

- A truth assignment is a function specifying the truth values for the variables in a formula.

Example (Truth assignment)

$$T = \{(x_1, 1), (x_2, 0), (x_3, 1)\}$$

- A truth assignment T satisfies a formula ϕ iff T makes ϕ true. We write this as $T \models \phi$.

SAT

$$\text{SAT} = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable boolean formula } \}$$

The SAT problem is this:

Input:	A boolean formula ϕ
Output:	Yes if ϕ is satisfiable. No, otherwise.

Boolean logic (cont.)

- A literal is either a boolean variable x or its negation $\neg x$.
- A clause is a boolean formula of the form $l_1 \vee \ldots \vee l_k$ where $k \in \mathrm{Z}^+$ and l_i is a literal.
- A formula is in conjunctive normal form (CNF) if it is of the form $c_1 \wedge \ldots c_k$ where $k \in \mathrm{Z}^+$ and c_i is a clause.
- A formula is in 3CNF if it is in conjunctive normal form and if each clause has exactly 3 pairwise unrelated literals.
- Two literals are related if they are identical or one is the complement of the other.

Example

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge x_4$ is in CNF
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (x_4 \vee \neg x_1 \vee x_2)$ is in 3CNF

Problems related to boolean logic

Example

$$
\begin{aligned}
\text{EVAL} &= \{\langle \phi, T\rangle \mid T \models \phi\} \\
\text{3CNF} &= \{\langle \phi\rangle \mid \phi \text{ is in 3-conjunctive normal form }\} \\
\text{3SAT} &= \{\langle \phi\rangle \mid \phi \text{ is a satisfiable formula in 3CNF}\} \\
\text{2SAT} &= \{\langle \phi\rangle \mid \phi \text{ is a satisfiable formula in 2CNF}\}
\end{aligned}
$$

Which ones are in P?
Which ones seem harder? Are they easy to verify?

Outline

1. Example decision problems
2. Complexity class NP
    - Examples
3. P = NP???
    - NP-Completeness
    - Polynomial time reducibility
    - Proving a problem NP-complete

Some languages in NP

1. CLIQUE $= \{\langle G, k\rangle \mid G$ is a undirected graph that has a k-clique $\}$.

   CLIQUE $\in$ NP

2. SUBSET-SUM $= \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}$ and for some $\{y_1, \ldots, y_n\} \subseteq \{x_1, \ldots, x_k\}$, we have $\sum y_i = t\}$.

   SUBSET-SUM $\in$ NP

Question: What about $\overline{\text{CLIQUE}}$ and $\overline{\text{SUBSET-SUM}}$??

Consider also the languages in the long lists we saw earlier.

Summary

P	=	class of languages where membership can be decided quickly
NP	=	class of languages where membership can be decided quickly by a nondeterministic Turing machine
NP	=	class of languages where membership can be verified quickly

Some problems, e.g. HAMPATH and CLIQUE, are in NP but are not known to be in P.

The big open question:

$$\mathrm{P} = \mathrm{NP}???$$

[e.g. can problems like HAMPATH and CLIQUE be solved in deterministic polynomial time?]

Outline

P = NP??

For now, we only know that

$$\begin{aligned} \mathrm{P} &\subseteq \mathrm{NP} \\ \mathrm{NP} &\subseteq \mathrm{EXP} = \bigcup_k \mathrm{TIME}(2^{n^k}) \,. \end{aligned}$$

But we do not know whether

$$\mathrm{NP} \subseteq \mathrm{P} = \bigcup_k \mathrm{TIME}(n^k) \text{ ???}$$

Surprisingly, Cook and Levin found that SAT is among the hardest problems in NP

in the sense that, if we can solve SAT in deterministic polynomial time, then we can solve all of the problems in NP!

i.e. if SAT $\in$ P, then NP $\subseteq$ P, which means that NP = P.

SAT is among the hardest problems in NP

Theorem (Cook-Levin)

SAT *is NP-complete.*

And we can show that the following theorem is true.

Theorem

Suppose A is NP-complete. Then, we have that

$$\textit{if } A \in \mathrm{P} \textit{ then } \mathrm{P} = \mathrm{NP} \ .$$

It turns out that there are many other NP-complete problems.

If theoreticians can solve any one of these NP-complete problems in deterministic polynomial time, then we would have P = NP.

If a practitioner working on a problem cannot solve the problem in deterministic polynomial time, then he can instead try to show that the problem is NP-complete.

NP-Completeness: Definition

Definition

A language L is NP-complete iff it satisfies two conditions:

1. $L \in \text{NP}$
2. L is NP-hard.

Definition

A language L NP-hard iff every $L' \in \text{NP}$ is polynomial time reducible to L.

If we can solve L in a short time, we can solve all the problems in NP in a short time. So P would be equal to NP!

Polynomial time reducibility: Definitions

The reduction that we saw can be formalized explicitly as a polynomial time reduction.

Definition

Language A is polynomial time reducible to language B, written $A \leq_P B$ iff there is a polynomial time reduction of A to B.

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial reduction of a language A to a language B iff f is polynomial computable and, for every $w \in \Sigma^*$,

- If $w \in A$, then $f(w) \in B$, and
- If $w \notin A$, then $f(w) \notin B$.

Properties we get from polynomial time reducibility

Let A, B be languages. Suppose $A \leq_P B$.

1. If $B \in \mathrm{P}$, then so is A.
2. If $A \notin \mathrm{P}$, then neither is B.

Also, these statements are true:

- $A \leq_P B$ iff $\overline{A} \leq_P \overline{B}$
- If $A \leq_P B$ and $B \leq_P C$, then $A \leq_P C$.
  (i.e. $\leq_P$ is a transitive relation.)

Questions: Is $\leq_P$ reflexive? Is it symmetric?

NP-complete problems are the hardest ones in NP

The theorem we saw earlier follows directly from polynomial time reducibility.

Theorem

Suppose A is NP-complete. Then, we have that

$$\textit{if } A \in \mathrm{P} \textit{ then } \mathrm{P} = \mathrm{NP} .$$

The theorem says that, if there is an NP-complete problem that ends up being easily solvable (i.e. solvable in polynomial time), then all the problems in NP become easily solvable.

So there are no problems in NP harder than NP-complete problems.

How to prove a problem NP-complete?

We reduce an NP-complete problem to L.

Theorem

Let $L \in \mathrm{NP}$. If C is NP-complete and $C \leq_P L$, then L is NP-complete.

This theorem is true because polynomial reducibility is transitive.

So to prove that a problem L is NP-complete, we prove that

1. $L \in \mathrm{NP}$, and that
2. there is a reduction from an NP-complete problem to L.

The first NP-complete problem

The problem that gets everything started is SAT.

In particular, Cook and Levin proved that any language in NP reduces to SAT.

Theorem (Cook-Levin)

SAT *is NP-complete.*

[As you might expect, the proof is somewhat complicated.]

So we "grow" our list of NP-complete problems starting from SAT.

3SAT is NP-complete

Theorem

3SAT *is NP-complete.*

To prove this theorem, we proceed as follows:

1. Show that 3SAT $\in$ NP
2. Show that SAT $\leq_P$ 3SAT
   1. Specify a reduction f
   2. Prove that f is polynomial time computable
   3. Prove that $\forall w, w \in$ SAT if and only if $f(w) \in$ 3SAT.

Proving other languages NP-complete

Usually, it is easy to show that a language is in NP. The tough part is often in showing that it is NP-hard.

1. 3SAT $\leq_P$ IS
2. IS $\leq_P$ CLIQUE
3. IS $\leq_P$ VC