

CS5371

Theory of Computation

Lecture 19: Complexity IV
(More on NP, NP-Complete)

Objectives

- More discussion on the class NP
- Cook-Levin Theorem

The Class NP revisited

- Recall that NP is the class of language that has a polynomial time verifier
- If a language L is in NP, there is a polynomial time TM V (verifier) such that
 - For each w in L , there is some string c such that $\langle w, c \rangle$ is accepted by V , and
 - For every w not in L , $\langle w, c \rangle$ is rejected by V for all c

The Class NP revisited (2)

The reason why V is called a verifier for L :

- if a string w is in L , it has at least one certificate c to present to V , allowing V to **verify** that it is in L
- if a string w is not in L , it will not have any certificate c to present to V , so that V will never make a mistake to **verify** that it is in L

V is actually like a judge in the old days: It tends to say you are guilty, unless you can prove yourself to be innocent!

Examples (HAMPATH)

- A trivial certificate for a string $\langle G \rangle$ in **HAMPATH** (i.e., for a graph G to be Hamiltonian) is the order of vertices visited in a Hamiltonian path in G
 - We can find a verifier V that uses this trivial certificate to prove $\langle G \rangle$ is in **HAMPATH** in polynomial time (in terms of the length of $\langle G \rangle$)
- Also, using this V , a non-Hamiltonian graph can never find a certificate to “prove” that it is Hamiltonian

Examples (COMPOSITE)

- The certificate for a string $\langle x \rangle$ in **COMPOSITE** (I.e., the number x is a composite number) is a factor of x between 2 to $x-1$
 - The corresponding verifier can use the certificate to prove $\langle x \rangle$ is in **COMPOSITE** in polynomial time (in terms of $\log x$ --- the length of $\langle x \rangle$)
- Also, when a number not composite, no certificate can **fool** this verifier

Properties of NP

Theorem: A language is in NP if and only if it is decided by a NTM that runs in polynomial time.

Prove idea: We show how to convert a verifier into a NTM and vice versa...

Properties of NP (2)

Proof: (\Rightarrow) Let A be a language in NP, so that it can be verified by some polynomial time verifier V . Let n^k be the running time of V . We create a polynomial time NTM N that decides A as follows:

N = "On input w ,

1. Select a string c of length at most n^k
2. Run V on $\langle w, c \rangle$
3. If V accepts, accept. Else, reject."

Properties of NP (3)

Proof: (\Leftarrow) Let A be a language decided by some polynomial time NTM N . We construct a polynomial time verifier V as follows:

V = "On input $\langle w, c \rangle$,

1. Simulate N on w , treating c as the description of the non-deterministic choice of N at each step
2. If this branch of computation in N accepts, accept. Else, reject."

Properties of NP (4)

Definition: $\text{NTIME}(t(n))$ = the set of languages that can be decided by an NTM that runs in $O(t(n))$ time

Based on the above definition and the previous theorem, we have:

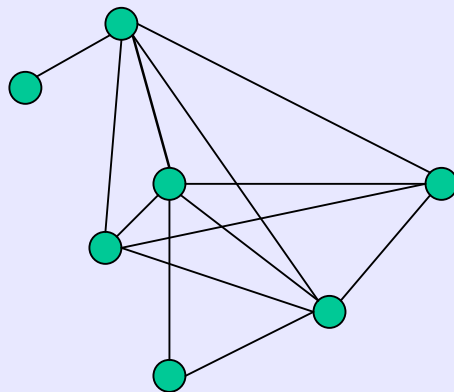
Corollary: $\text{NP} = \bigcup_k \text{NTIME}(n^k)$

More Examples of NP

Definition: A **clique** is a subgraph G' in an undirected graph G such that every two nodes in G' are connected.

Definition: A **k-clique** is a clique that contains k nodes.

E.g.,



Can you find a 5-clique here?

More Examples in NP (2)

Let **CLIQUE** be the language

$\{ \langle G, k \rangle \mid G \text{ is a graph with a } k\text{-clique} \}$

Theorem: **CLIQUE** is in NP.

How to prove??

CLIQUE is in NP

Proof 1 (using DTM verifier): What should be the certificate that a graph can prove itself is a k -clique??

The following is the verifier for **CLIQUE**:

$V =$ "On input $\langle G, k, c \rangle$

1. Test if c is a set of k nodes in G
2. Test whether every two nodes in c are connected
3. If both pass, **accept**. Else, **reject**."

CLIQUE is in NP (2)

Proof 2 (using NTM decider): What should be the "guess" made by the NTM in order to check if G contains a k -clique??

The NTM below is one that decides **CLIQUE**:

$N =$ "On input $\langle G, k \rangle$

1. Non-deterministically select a subset c containing k nodes of G
2. Check if every two nodes in c are connected
3. If yes, **accept**. Else, **reject**."

Another Examples in NP

Let **SUBSET-SUM** be the language

$\{ \langle S, t \rangle \mid S \text{ is a set of integers such that} \\ \text{a subset of } S \text{ adds up to } t \}$

Theorem: **SUBSET-SUM** is in NP.

How to prove??

SUBSET-SUM is in NP

Proof 1 (using DTM verifier): What should be the certificate that S can prove itself has a subset that adds up to t ??

The verifier below is one for SUBSET-SUM:

$V =$ "On input $\langle S, t, c \rangle$

1. Test if c is a set of numbers in S
2. Test if the numbers in c adds up to t
3. If both pass, accept. Else, reject."

SUBSET-SUM is in NP (2)

Proof 2 (using NTM decider): What should be the non-deterministic guess made by the NTM in order to check if S contains a subset that adds up to t ??

The NTM below is one that decides

SUBSET-SUM:

N = "On input $\langle S, t \rangle$

1. Non-deterministically find a subset c of S
2. Check if the numbers in c adds up to t
3. If yes, **accept**. Else, **reject**."

P versus NP

Roughly speaking:

P = the class of language that can be
decided "quickly"

NP = the class of language that can be
verified "quickly"

- The power of the polynomial time NTM decider seems to be much greater than the polynomial time DTM decider...

P versus NP (2)

- ... Unfortunately, so far, nobody can tell whether $P = NP$, or $P \neq NP$, is true
- A general belief (which may not be true) is $P \neq NP$ because people have input a lot of effort to find polynomial time algorithms for certain problems in NP, but fail
- What we can conclude safely so far is:

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

NP-Completeness

In the early 1970s, Stephen Cook and Leonid Levin (separately) discovered that:
Some languages in **NP**, if any of them are decidable by a DTM in polynomial time, will imply **ALL** problems in **NP** can be decided by a DTM in polynomial time
That is, they discovered some language **L**, such that if **L** is in **P**, then **P** = **NP**

NP-Completeness (2)

These problems are called **NP-complete** problems, and they form a class **NP-C** (We shall give formal definition later)

The first NP-complete problem we present is called the **satisfiability problem**

Satisfiability Problem

Definition: A variable v is called a **Boolean variable** if it has a value either 1 (TRUE) or 0 (FALSE)

Definition: The **Boolean operations** \wedge , \vee , \neg , are defined as follows:

$$0 \wedge 0 = 0, \quad 0 \vee 0 = 0, \quad \neg 0 = 1$$

$$0 \wedge 1 = 0, \quad 0 \vee 1 = 1, \quad \neg 1 = 0$$

$$1 \wedge 0 = 0, \quad 1 \vee 0 = 1,$$

$$1 \wedge 1 = 1, \quad 1 \vee 1 = 1$$

Satisfiability Problem (2)

Definition: A **Boolean formula** is an expression involving Boolean variables and operations

E.g., The following is a Boolean formula:

$$F = (\neg x \wedge y) \vee (x \wedge \neg z)$$

Satisfiability Problem (2)

Definition: A Boolean formula is **satisfiable** if some assignments of 0 and 1 to the variables makes the value of the formula equal to 1

E.g., The previous Boolean formula

$$F = (\neg x \wedge y) \vee (x \wedge \neg z)$$

is satisfiable because if we set $x = 0$, $y = 1$, and $z = 1$, the value of F becomes 1

Cook-Levin Theorem

Let SAT be the language

$\{ \langle F \rangle \mid F \text{ is a satisfiable Boolean formula} \}$

Theorem: SAT is P if and only if $P = NP$

Next Time

- Polynomial Time Reducibility
- Prove Cook-Levin Theorem
- Proving other problems to be NP-Complete