

# CS 39

## Theory of Computing

Introduction to NP-completeness  
Amit Chakrabarti

## Recap: P and NP

- $P = \{L \subseteq \Sigma^* : L \text{ is decided by a TM in polynomial time}\}$
- $NP = \{L \subseteq \Sigma^* : L \text{ is decided by a NDTM in polynomial time}\}$
- In CS 25 you (essentially) learnt techniques to show that various languages  $\in P$ .
- How do we show that a language  $\in NP$ ?

## The Hamiltonian Path Problem

- Input: A graph  $G = (V, E)$
- Question: Does  $G$  have a Hamiltonian path?
- Definition: A Hamiltonian path of  $G$  is a path that covers all vertices of  $G$ .
- To turn this into a language, define  
$$\text{HAMPATH} = \{\langle G \rangle : G \text{ is a graph that has a Hamiltonian path}\}.$$

## The Vertex Cover Problem

- Input: A graph  $G = (V, E)$  and an integer  $k > 0$
- Q: Does  $G$  have a vertex cover of size  $\leq k$ ?
- Definition: A vertex cover of  $G$  is a subset of  $V$  that covers (i.e., “touches”) every edge in  $E$ .
- To turn this into a language, define  
$$\text{VC} = \{\langle G, k \rangle : G \text{ is a graph that has a vertex cover of size } \leq k\}.$$

## Proof that HAMPATH $\in$ NP

- “On input  $\langle G \rangle$ , where  $G = (V, E)$  is a graph:
  1. Let  $n = |V|$ .
  2. Guess a permutation  $v_1, v_2, \dots, v_n$  of  $V$ .
  3. For  $i = 1$  to  $(n-1)$ :
    - 3.1. If  $\{v_i, v_{i+1}\} \notin E$ , then REJECT.
  4. ACCEPT.”
- Clearly polynomial time.
- Uses nondeterminism in step 2.

## Proof that VC $\in$ NP

- “On input  $\langle G, k \rangle$ , where  $G = (V, E)$ ... :
  1. Guess a subset  $C = \{v_1, v_2, \dots, v_k\}$  of  $V$ .
  2. For each edge  $\{u, v\} \in E$ :
    - 3.1. If  $u \notin C$  and  $v \notin C$ , then REJECT.
  3. ACCEPT.”
- Clearly polynomial time.
- Uses nondeterminism in step 1.

## Do we *need* to guess?

- We showed that HAMPATH, VC  $\in$  NP.
- Their (nondeterministic) algorithms used the power to guess in a crucial way.
- Enumerating all guesses
  - all permutations, in case of HAMPATH
  - all  $k$ -sized subsets, in case of VCcould take *exponential* time (w.r.t. input size).

## More examples of NP problems

- SATISFIABILITY, a.k.a. SAT:
  - Input: A *formula*, i.e., the AND of a set of Boolean *clauses*, e.g.
    - $x_1 \vee \neg x_2 \vee x_3$
    - $\neg x_1 \vee \neg x_2$
    - $x_4 \vee x_2 \vee x_5 \vee \neg x_7 \vee x_1$
  - Question: Is the formula satisfiable? I.e., is there a TRUE/FALSE assignment to the  $x_i$ s that makes the formula true?
- Note: *Every* clause must be satisfied.

## Proof that SAT $\in$ NP

- SAT =  $\{\langle\phi\rangle: \phi \text{ is a satisfiable formula}\}$
- “On input  $\langle\phi\rangle$ ,
  1. Guess a Boolean value (TRUE/FALSE) for each variable that occurs in  $\phi$ .
  2. If the guessed values satisfy all the clauses of  $\phi$ , then ACCEPT, else REJECT.”
- Deterministic algorithm? Enumerating all guesses could take  $2^{O(n)}$  time, where  $n = |\langle\phi\rangle|$ .

## More examples of NP problems

- The SUBSET-SUM problem:
  - Input: A finite set of integers  $S$  and a target integer  $t$ .
  - Question: Is there a subset  $T \subseteq S$  such that the sum of the elements of  $T$  equals  $t$ ?
- Again, clearly in NP: just guess a subset and verify that it sums to  $t$ .

## Polynomial-time reductions

- We’ve now seen several problems that are in NP but don’t seem to be in P:  
HAMPATH, VC, SAT, SUBSET-SUM
- We shall see: if we could somehow solve *one* of these problems in P-time, we could solve *all* of them in P-time.
- How? Via P-time reductions.
  - i.e., reductions that run in polynomial time.

## Mapping reductions

Our P-time reductions will be *mapping reductions*

Reduction  $A \rightarrow B$  will look like

“On input  $x$ :

- Perform some computation to produce  $y = f(x)$
- Output  $y$ ”

Essential property of  $f$ :  $x \in A \Leftrightarrow f(x) \in B$

The computation (i.e.,  $f$ ) “maps” A to B

## NP-completeness

- A language  $L$  is said to be *NP-complete* if
  1.  $L \in \text{NP}$
  2. Every language in NP can be P-time reduced to  $L$ .
- In other words, the power to solve  $L$  gives us the power to solve *everything* in NP!
  - Here “solve” means “solve in polynomial time.”
- In still other words, if  $L \in \text{P}$  then  $\text{P} = \text{NP}$ .

## What it means to be NP-complete

- Suppose we’ve proven (somehow) that a language  $L$  is NP-complete.
- This *suggests* that  $L$  can’t be decided in P-time.
  - Because, if  $L$  *could* be decided thus, then so could *every* problem in NP...
  - ...such as these one thousand problems that generations of brilliant computer scientists have been unable to solve...
- Suggests, but *does not prove*.

## How to prove NP-completeness

- A language  $L$  is said to be *NP-complete* if
  - (1)  $L \in \text{NP}$
  - (2) Every language in NP can be P-time reduced to  $L$ .
- Suppose we’ve proven (somehow) that SAT is NP-complete. We wish to prove that VC is, too.
- Prove (1). For (2), just reduce SAT to VC!
  - Any NP language  $\rightarrow \text{SAT} \rightarrow \text{VC}$

## NP-completeness of VC

- We’ve already proven (1)  $\text{VC} \in \text{NP}$
- For (2), we’ll use several steps:
  - First, we reduce SAT to 3SAT.
  - Then, we reduce 3SAT to IND-SET.
  - Finally, we reduce IND-SET to VC.
  - Each of these reductions will run in polynomial time.

## SAT $\rightarrow$ 3SAT

- 3SAT is just like SAT, except that each clause in the formula is required to have exactly 3 literals.
  - $x_1 \vee \neg x_2 \vee x_3$
  - $\neg x_1 \vee \neg x_2 \vee x_5$
  - $x_4 \vee x_2 \vee x_5$
- To convert an arbitrary formula into this form, need to deal with
  - clauses that have only 1 or 2 literals,
  - clauses that have 4 or more literals.

## SAT $\rightarrow$ 3SAT

- Clauses with too few literals
  - Replicate literals to bring the number up to 3,
  - E.g.,  $(\neg x_1 \vee x_5) \rightarrow (\neg x_1 \vee \neg x_1 \vee x_5)$
  - and  $(x_3) \rightarrow (x_3 \vee x_3 \vee x_3)$
- Clauses with too many literals
  - Chaining: split into multiple clauses, using new “link” literals.
  - E.g.,  $(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee z) \wedge (\neg z \vee x_3 \vee x_4)$
  - Replace  $(x_1 \vee \dots \vee x_k)$  with  $(k-2)$  new clauses:  
 $(x_1 \vee x_2 \vee z_3) \wedge (\neg z_3 \vee x_3 \vee z_4) \wedge (\neg z_4 \vee x_4 \vee z_5) \wedge \dots$   
 $\dots \wedge (\neg z_{k-1} \vee x_{k-1} \vee x_k)$  [Like converting a CFG into CNF]
- Check that all this can be done in poly time.

## 3SAT $\rightarrow$ IND-SET

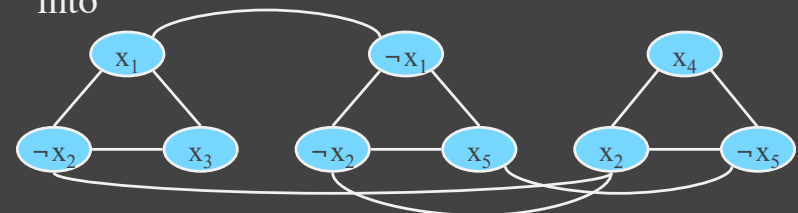
- The IND-SET problem asks whether a given input graph has an *independent set* of a given size.
  - An *independent set* is a set of vertices such that no two of them are adjacent.
- Thus, the *larger* an independent set, the more interesting it is. Can we find the *largest*?
- Decision (yes/no) version: Given  $G$  and  $k$ , does  $G$  have an independent set of size  $\geq k$ ?

## 3SAT $\rightarrow$ IND-SET

- Must convert 3cnf-formula  $\phi$  into  $G$  and  $k$ , s.t.
  - If  $\phi$  satisfiable, then  $G$  has an i.s. of size  $k$ .
  - If  $\phi$  unsatisfiable, then  $G$  doesn't have i.s. of size  $k$ .
- Idea: turn

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

into



## 3SAT $\rightarrow$ IND-SET

- Formally, “On input  $\langle \phi \rangle$ :
  - Let  $C_1, \dots, C_k$  be the clauses of  $\phi$ .
  - Create a  $3k$ -vertex graph  $G$  where each vertex corresponds to a literal in some  $C_i$  as follows:
  - Draw  $k$  disjoint triangles, one per clause.
  - Then add extra edges connecting each pair of contradicting literals.
  - Output  $\langle G, k \rangle$ .”
- Why does this work? Prove it!

## IND-SET $\rightarrow$ VC

- Theorem: Suppose  $G$  has  $n$  vertices. Then  $G$  has an independent set of size  $k$  iff  $G$  has a vertex cover of size  $(n - k)$ .
  - Proof sketch: The vertices *not* in an independent set form a vertex cover.
- This theorem leads to a very simple reduction:
  - “On input  $\langle G, k \rangle$ ”
    1. Let  $n =$  number of vertices of  $G$ .
    2. Output  $\langle G, n-k \rangle$ .”

## Recap

- We have shown these reductions:  
SAT  $\rightarrow$  3SAT  $\rightarrow$  IND-SET  $\rightarrow$  VC
- Therefore, if we could show SAT is NP-complete
  - we would have shown that 3SAT is NP-complete.
  - we would have shown that IND-SET is NP-complete.
  - we would have shown that VC is NP-complete.
- Eventually: **Cook-Levin theorem**, which proves from scratch that SAT is NP-complete.

## Very important reading assignment

- Read Sipser, pages 248-253.
- Read Sipser, section 7.5 completely.
  - There you will find proofs that HAMPATH and SUBSET-SUM are NP-complete.
  - We will not be doing these proofs in class, but you are responsible for knowing and understanding them.