CS372
FORMAL LANGUAGES & THE THEORY OF COMPUTATION

Dr.Nguyen Thi Thu Huong
Department of Computer Science
Phone: 38696121, Mobi: 0903253796
Email: huongnt@soict.hust.edu.vn,
huong.nguyenthithu@hust.edu.vn

Unit 11
Post’s Correspondence Problem

Post's Correspondence Problem (PCP)

AN INSTANCE OF THE PCP

A PCP instance over Σ is a finite collection P of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\}$$

where for all i, $1 \leq i \leq k$, $t_i, b_i \in \Sigma^+$.

Given a PCP instance P, a match is a nonempty sequence

$$i_1, i_2, \ldots, i_e$$

of numbers from $\{1, 2, \ldots, k\}$ (with repetition) such that $t_{i_1} t_{i_2} \cdots t_{i_e} = b_{i_1} b_{i_2} \cdots b_{i_e}$

A match of PCP

Alphabet: $\Sigma = \{a, b, c\}$

a	*c*	*ba*	*a*	*acb*
ac	*ba*	*a*	*ac*	*b*

Concatenation of strings on top: a c ba a acb

Concatenation of strings at bottom: ac ba a ac b

Post's Correspondence Problem (PCP)

QUESTION:

Does a given PCP instance P have a match?

LANGUAGE FORMULATION:

$PCP = \{(P) \mid P$ is a PCP instance and it has a match$\}$

THEOREM 11.2

PCP is undecidable.

Proof: By reduction using computation histories. If PCP is decidable then so is A_{TM}. That is, if PCP has a match, then M accepts w.

A “yes” instance of PCP

Alphabet: $\Sigma = \{0,1\}$

	T	B
i	t_i	b_i
1	11	1
2	1	111
3	0111	10
4	10	0

This instance of PCP has a match of dominos 1, 3, 2, 2, 4:

$$t_1t_3t_2t_2t_4 = b_1b_3b_2b_2b_4 = 1101111110$$

A “no” instance of PCP

Alphabet: $\Sigma = \{0,1\}$

	T	B
i	t_i	b_i
1	10	01
2	011	100
3	101	010

Why?

PCP – The Structure of the Undecidability Proof

The reduction works in two steps:

We reduce A_{TM} to Modified PCP (MPCP).

We reduce MPCP to PCP.

MPCP as a language problem

$MPCP = \{(P) \mid P$ is a PCP instance and it has a match which starts with index 1$\}$

So the solution to MPCP starts with the domino $\frac{t_1}{b_1}$. We later remove this restriction in the second part of the proof.

We also assume that the decider for M never moves its head to the left of the input w.

Mapping Reducibility

Definition

Let $A, B \subseteq \Sigma^*$. We say that language A is mapping reducible to language B, written $A <_m B$, if and only if

1. There is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that
2. For every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.

The function f is called a reduction of A to B.

THEOREM 11.3

If $A <_m B$ and B is decidable, then A is decidable.

Proof

Let M be a decider for B and f be a mapping from A to B.

Then N decides A. $N =$ "On input w

1. Compute $f(w)$
2. Run M on input $f(w)$ and output whatever M outputs."

If $A <_m B$ and A is undecidable, then B is undecidable.

Summary of Mapping Reducibility Results

Summary of Theorems

Assume that $A <_m B$. Then

1. If B is decidable then A is decidable.
2. If A is undecidable then B is undecidable.
3. If B is Turing-recognizable then A is Turing-recognizable.
4. If A is not Turing-recognizable then B is not Turing-recognizable.
5. $\overline{A} <_m \overline{B}$

Useful observation:

- Suppose you can show $A_{TM} <_m \overline{B}$
- This means $\overline{A_{TM}} <_m B$
- Since $\overline{A_{TM}}$ is Turing-unrecognizable then B is Turing-unrecognizable.

Example of Use

Theorem

$EQ_{TM} = \{(M_1, M_2) \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing recognizable nor co-Turing-recognizable.

Proof Idea

We show

- $\overline{A_{TM}} <_m EQ_{TM}$
- $\overline{A_{TM}} <_m \overline{EQ_{TM}}$
- These then imply the theorem.

Example of Use

Proof for $A_{TM} <_m EQ_{TM}$

We show $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$) with the following g:

G = "On input (M, w) where M is a TM and w is a string:

1. Construct the following two machines M_1 and M_2
   M_1 = "On any input:
   1. Accept"

   M_2 = "On any input:
   1. Run M on w. If it accepts, *accept*."
2. Output (M_1, M_2)."

- M_1 accepts everything.
  - If M accepts w then M_2 accepts everything. So M_1 and M_2 are equivalent.
  - If M does not accept w then M_2 accepts nothing. So M_1 and M_2 are not equivalent.
- So $A_{TM} <_m EQ_{TM}$ (and hence $\overline{A_{TM}} <_m \overline{EQ_{TM}}$)

Summary of Reducibility

We know that language A is undecidable. By reducing A to B we want to show that the language B is also undecidable.

1. Assume that we have a decider M_B for B.
2. Using M_B we construct a decider M_A for the language A:

$M_A =$ "On input (I_A)

1. Algorithmically construct an input (I_B) for M_B, such that

   a) Either

   If $(I_A) \in A$ then $(I_B) \in B$
   If $(I_A) \notin A$ then $(I_B) \notin B$

   b) or

   If $(I_A) \in A$ then $(I_B) \notin B$
   If $(I_A) \notin A$ then $(I_B) \in B$

2. Run the decider M_B on (I_B) for M_B
   Case a): M_A accepts if M_B accepts, and rejects if M_B rejects Case b): M_A rejects if M_B accepts, and accepts if M_B reject.

3. We know M_A can not exist so M_B can not exist.
4. B is undecidable.

Computable Functions

Idea

Turing Machines can also compute function $f : \Sigma^* \longrightarrow \Sigma^*$.

Computable Function

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if and only if there exists a TM M_f, which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Examples:

- Let $f(w) \overset{def}{=} ww$ be a function. Then f is computable.
- Let $f((n_1, n_2)) \overset{def}{=} (n)$ where n_1 and n_2 are integers and $n = n_1 * n_2$. Then f is computable.

Unit 12
Time Complexity

Complexity Theory

Complexity Theory aims to make general conclusions of the resource requirements of decidable problems (languages).

We only consider decidable languages and deciders. Our computational model is a Turing Machine.

- Time: the number of computation steps a TM machine makes to decide on an input of size *n*.
- Space: the maximum number of tape cells a TM machine takes to decide on a input of size *n*.

Motivation

- How much time (or how many steps) does a single tape TM take to decide $A = \{0^k 1^k \mid k \geq 0\}$?

M = "On input *w*:

1. Scan the tape and *reject* if *w* is not of the form 0^*1^*.
2. Repeat if both 0s and 1s remain on the tape.
3. Scan across the tape crossing off one 0 and one 1.
4. If all 0's are crossed and some 1's left, or all 1's crossed and some 0's left, then *reject*; else *accept*.

QUESTION

How many steps does *M* take on an input *w* of length *n*?

ANSWER (WORST-CASE)

The number of steps *M* takes n^2.

Some Notions

- The number of steps in measured as a function of *n* **- the size of the string representing the input.**
- In worst-case analysis, we consider the longest running time of all inputs of length *n*.
- In average-case analysis, we consider the average of the running times of all inputs of length *n*.

TIME COMPLEXITY

Let *M* be a deterministic TM that halts on all inputs. The time complexity of *M* is the function $f : \mathbb{N} \longrightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that *M* uses on any input of length *n*. If $f(n)$ is the running time of *M* we say

- *M* runs in time $f(n)$
- *M* is an $f(n)$-time TM.

Asymptotic Analysis

- We seek to understand the running time when the input is “large”.
- Hence we use an asymptotic notation or big-O notation to characterize the behaviour of $f(n)$ when n is large.
- The exact value running time function is not terribly important. What is important is how $f(n)$ grows as a function of n, for large n. Differences of a constant factor are not important.

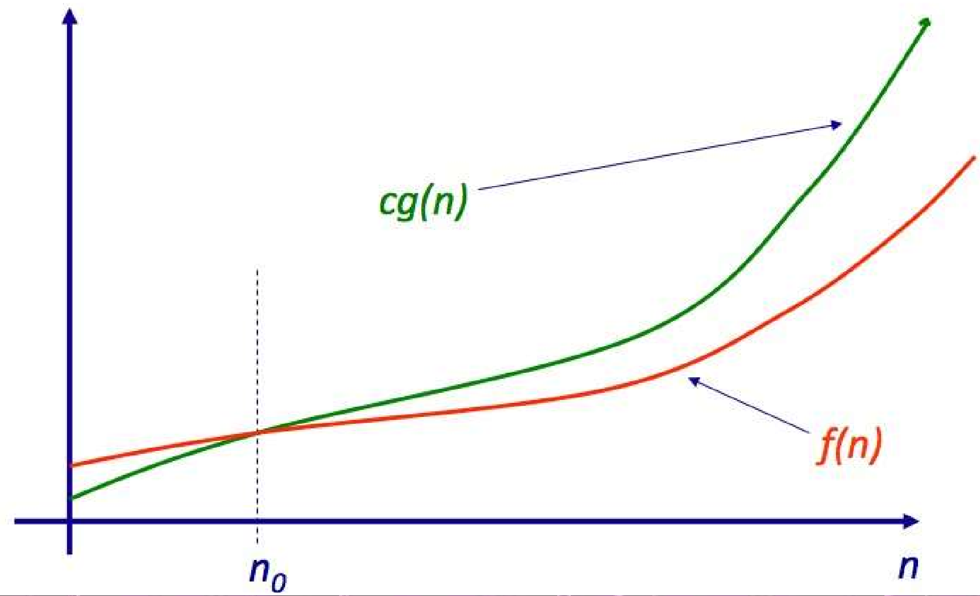
Asymptotic Upper Bound

DEFINITION – ASYMPTOTIC UPPER BOUND

Let R^+ be the set of nonnegative real numbers. Let f and g be functions $f, g : N \longrightarrow R^+$. We say $f(n) = O(g(n))$, if there are positive integers c and n_0, such that for every $n \geq n_0$

$$f(n) \leq c\, g(n).$$

$g(n)$ is an asymptotic upper bound.



Complexity Classes

DEFINITION – TIME COMPLEXITY CLASS TIME($t\,(n)$)

Let $t : \mathrm{N} \longrightarrow \mathrm{R}^+$ be a function.
TIME($t(n)$) = $\{L(M) \mid M$ is a decider running in time $O(t(n))\}$

TIME($t(n)$) is the class (collection) of languages that are decidable by TMs, running in time $O(t(n))$.

$\mathrm{TIME}(n) \subset \mathrm{TIME}(n^2) \subset \mathrm{TIME}(n^3) \subset \ldots \subset \mathrm{TIME}(2^n) \subset \ldots$

Examples:

$\{0^k 1^k \mid k \geq 0\} \in \mathrm{TIME}(n^2)$

$\{0^k 1^k \mid k \geq 0\} \in \mathrm{TIME}(n \log n)$

$\{w\#w \mid w \in \{0, 1\}^*\} \in \mathrm{TIME}(n^2)$

$\{0^k 1^k \mid k \geq 0\} \in \text{TIME}(n \log n)$

M = "On input w:

1. Scan the tape and *reject* if w is not of the form 0^*1^*.
2. Repeat as long as some 0s and some 1s remain on the tape.
   - Scan across the tape, checking whether the total number of 0s and 1s is even or odd. *Reject* if it is odd.
   - Scan across the tape, crossing off every other 0 starting with the first 0, and every other 1, starting with the first 1.
3. If no 0's and no 1's remain on the tape, *accept*. Otherwise, *reject*.

Steps 2 take $O(n)$ time.

Step 2 is repeated at most $1 + \log_2 n$ times. (why?)

Total time is $O(n \log n)$.

Hence, $\{0^k 1^k \mid k \geq 0\} \in \text{TIME}(n \log n)$.

However, $\{0^k 1^k \mid k \geq 0\}$ is decidable on a 2-tape TM in time $O(n)$

(How ?)

The Class P

DEFINITION

P is the class of languages that are decidable in polynomial time on a deterministic single-tape TM.

$$P = \bigcup_k \mathrm{TIME}(n^k).$$

The class P is important for two main reasons:

1. P is robust: The class remains invariant for all models of computation that are polynomially equivalent to deterministic single-tape TMs.
2. P (roughly) corresponds to the class of problems that are realistically solvable on a computer.

Even though the exponents can be large (though most useful algorithms have “low” exponents), the class P provides a reasonable definition of practical solvability.

Example of Problems in P

THEOREM

PATH = $\{(G, s, t) \mid G$ is a directed graph with n nodes that has a path from s to $t\}$ $\in$ P.

PROOF

M = "On input (G, s, t)

1. Place a mark on s.
2. Repeat 3 until no new nodes are marked
3. Scan edges of G. If (a, b) is an edge and a is marked and b is unmarked, mark b.
4. If t is marked, *accept* else *reject*."

- Steps 1 and 4 are executed once
  - Each takes at most $O(n)$ time on a TM.
- Step 3 is executed at most n times
  - Each execution takes at most $O(n^2)$ steps ($\propto$ number of edges)
- Total execution time is thus a polynomial in n.

Example of Problems in P

THEOREM

$A_{CFG} \in \mathrm{P}$

PROOF.

The CYK algorithm decides A_{CFG} in polynomial time.

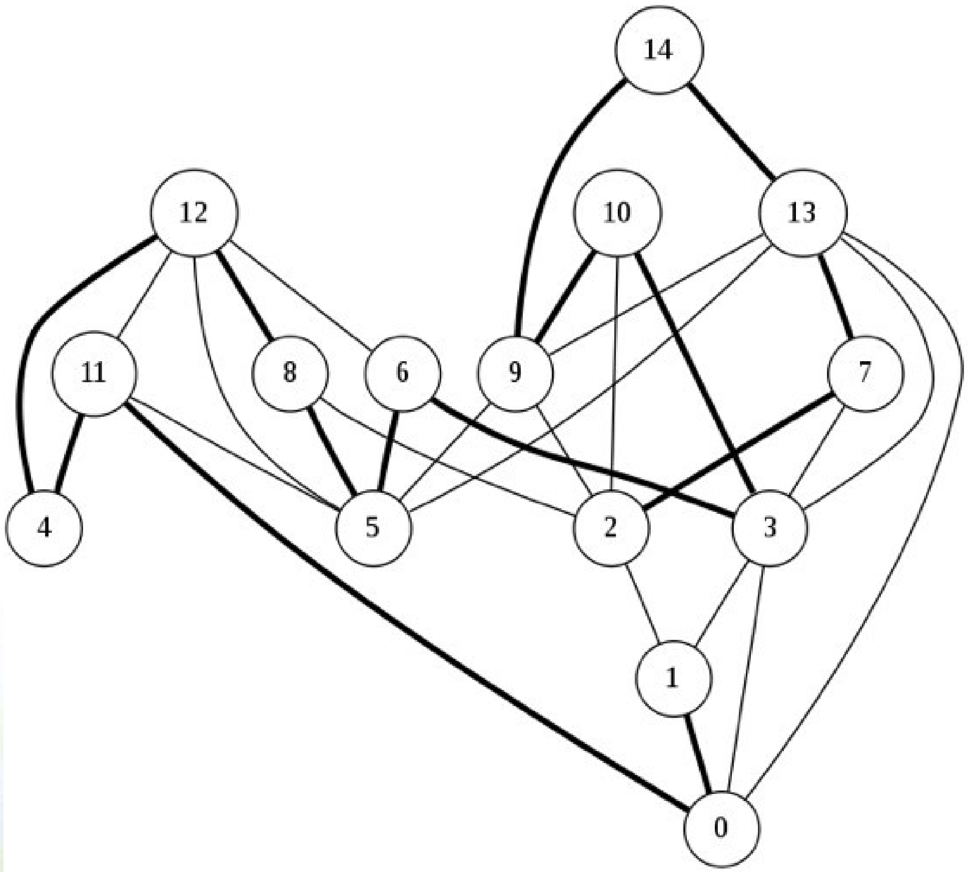
The Class NP

- For some problems, even though there is a exponentially large search space of solutions (e.g., for the path problem), we can avoid a brute force solution and get a polynomial-time algorithm.
- For some problems, it is not possible to avoid a brute force solution and such problems have so far resisted a polynomial time solution.
- We may not yet know the principles that would lead to a polynomial time algorithm, or they may be “intrinsically difficult.”
- How can we characterize such problems?

The Hamiltonian Path Problem

DEFINITION – HAMILTONIAN PATH

A Hamiltonian path in a directed graph *G* is a directed path that goes through each node exactly once.



The Hamiltonian Path Problem

HAMILTONIAN PATH PROBLEM

HAMPATH = {(*G*, *s*, *t*) | *G* is a directed graph with a Hamiltonian path from *s* to *t*}.

- We can easily obtain an exponential time algorithm with a brute force approach.

  Generate all possible paths between *s* and *t* and check if all nodes appear on a path!

- The *HAMPATH* problem has a property called polynomial verifiability.

  If we can (magically) get a Hamiltonian path, we can verify that it is a Hamiltonian path, in polynomial time

- *Verifying* the existence of a Hamiltonian path is "easier" than *determining* its existence.

The Class NP

THE CLASS NP

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

- NP stands for nondeterministic polynomial time.
- Problems in NP are called NP-Problems.
- $P \subset (\subseteq?)$ NP.

The Class NP

THEOREM 12.2

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

PROOF IDEA

- We show polynomial time verifier $\Leftrightarrow$polynomial time decider TM.
  - NTM simulates the verifier by guessing a certificate.
  - The verifier simulates the NTM

PROOF: NTM GIVEN THE VERIFIER.

Let $A \in$NP. Let V be a verifier that runs in time $O(n^k)$. N decides A in nondeterministic polynomial time.

N = "On input w of length n

1. Nondeterministically select string c of length at most n^k.
2. Run V on input (w, c).
3. If V accepts, *accept*; otherwise *reject*."

The Class NP

DEFINITION

$\mathrm{NTIME}(t(n)) = \{L \mid L$ is a language decided by a $O(t(n))$ time nondeterministic TM.$\}$

COROLLARY

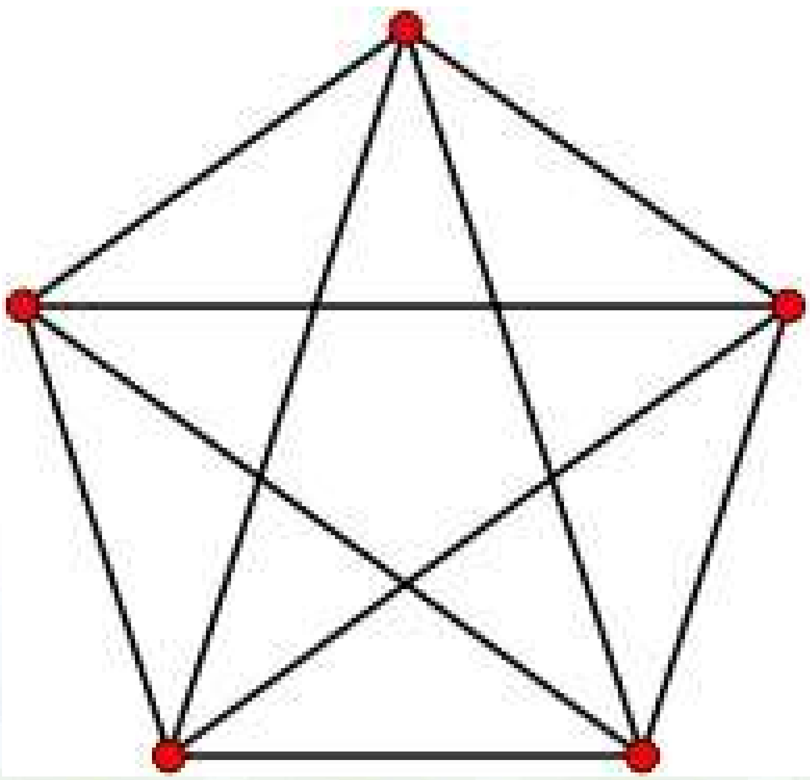
$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

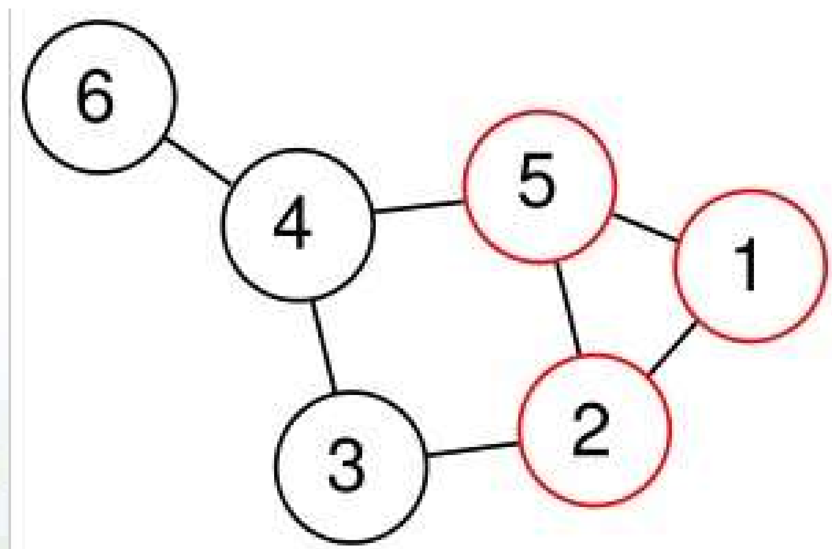
The clique problem

DEFINITION - CLIQUE

A clique in 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.





The Clique Problem

THEOREM 12.3

$CLIQUE = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\} \in NP$.

PROOF

The clique is the certificate.

V = "On input $((G, k), c)$:

1. Test whether c is a set of k nodes in G.
2. Test whether G has all edges connecting nodes in c.
3. If both pass, *accept*; otherwise *reject*."

- All steps take polynomial time

ALTERNATIVE PROOF

Use a NTM as a decider.

N = "On input (G, k):

1. Nondeterministically select a subset c of k nodes of G.
2. Test whether G has all edges connecting nodes in c.
3. If yes *accept*; otherwise *reject*."

The Class CoNP

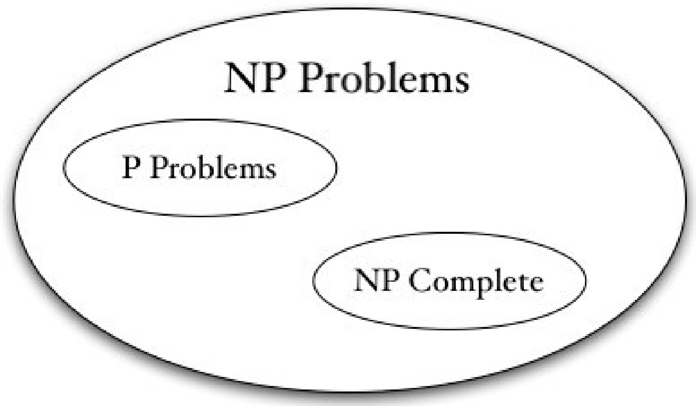
- It turns out $\overline{CLIQUE}$ or $\overline{SUBSET\text{-}SUM}$ are NOT in NP.
- Verifying something is NOT present seems to be more difficult than verifying it IS present.
- The class coNP contains all problems that are complements of languages in NP.
- We do not know if coNP $\neq$ NP.

Unit 13

NP Completeness

Summary of Complexity

- Time complexity: Big-O notation, asympotic complexity
- Simulation of multi-tape TMs with a single tape deterministic TM can be done with a polynomial slow-down.
- Simulation of nondeterministic TMs with a deterministic TM is exponentially slower.
- The Class P: The class of languages for which membership can be *decided* quickly.
- The Class NP: The class of languages for which membership can be *verified* quickly.



- We do not yet know if P = NP, or not.

NP Problems

- The best method known for solving languages in NP deterministically uses exponential time, that is

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

- It is not known whether NP is contained in a smaller deterministic time complexity class.

NP-complete Problems

- Cook and Levin in early 1970's showed that certain problems in NP were such that
  - If any of these problems had a deterministic polynomial-time algorithm, then
  - All problems in NP had deterministic polynomial-time algorithms.
- Such problems are called NP-complete problems.
- This is important for a number of reasons:
  1. If one is attempting to show that P/=NP, s/he may focus on an NP-complete problem and try to show that it needs more than a polynomial amount of time.
  2. If one is attempting to show that P=NP, s/he may focus on an NP-complete problem and try to come up with a polynomial time algorithm for it.
  3. One may avoid wasting searching for a nonexistent polynomial time algorithm to solve a particular problem, if one can show it reduces to an NP-complete problem

The Satisfiability Problem

DEFINITION – BOOLEAN VARIABLES

A boolean variable is a variable that can taken on values TRUE (1) and FALSE (0).

- We have Boolean operations of AND ($x \wedge y$), OR ($x \vee y$) and NOT ($\neg x$ or $\overline{x}$) on boolean variables.

AND	OR	NOT
$0 \wedge 0 = 0$	$0 \vee 0 = 0$	$\overline{0} = 1$
$0 \wedge 1 = 0$	$0 \vee 1 = 1$	$\overline{1} = 0$
$1 \wedge 0 = 0$	$1 \vee 0 = 1$	
$1 \wedge 1 = 1$	$1 \vee 1 = 1$	

The Satisfiability Problem

DEFINITION – BOOLEAN FORMULA

A Boolean formula is an expression involving Boolean variables and operations.

For example: $\varphi = (\overline{x} \wedge y) \vee (x \wedge \overline{z}) \vee (y \wedge z)$ is a Boolean formula.

DEFINITION – SATISFIABILITY

A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.

We say the assignment satisfies φ.

- What possible assignments satisfy the formula above?

DEFINITION – THE SATISFIABILITY PROBLEM

The satisfiability problem checks if a Boolean formula is satisfiable.

$$SAT = \{(\varphi) \mid \varphi \text{ is a satisfiable Boolean formula}\}$$

Polynomial Time Reducibility

DEFINITION – POLYNOMIAL TIME COMPUTABLE FUNCTION

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

DEFINITION – POLYNOMIAL TIME REDUCIBILITY

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

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

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

- To test whether $w \in A$ we use the reduction f to map w to $f(w)$ and test whether $f(w) \in B$.

Polynomial Time Reducibility

THEOREM 7.31

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

PROOF

- It takes polynomial time to reduce A to B.
- It takes polynomial time to decide B.

Variations on the Satisfiability Problem

- A literal is a Boolean variable or its negated version (x or $\overline{x}$).
- A clause is several literals connected with $\vee$ (OR), e.g., $(x_1 \vee \overline{x_2} \vee x_4)$.
- A Boolean formula is in conjuctive normal form (or is a cnf-formula) if it consists of several clauses connected with $\wedge$ (AND), e.g.

$$(x_1 \vee \overline{x_2} \vee x_4 \vee x_5) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3 \vee \overline{x_5})$$

- A cnf-formula is a 3cnf-formula if all clauses have 3 literals, e.g.

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_3 \vee \overline{x_5})$$

- $3SAT = \{(\varphi) \mid \varphi \text{ is a satisfiable 3cnf-formula}\}$.
  - In a satisfiable cnf-formula, each clause must contain at least one literal that is assigned 1.

An example reduction: Reducing 3*SAT* to *CLIQUE*

THEOREM 13.2

3*SAT* is polynomial time reducible to *CLIQUE* .

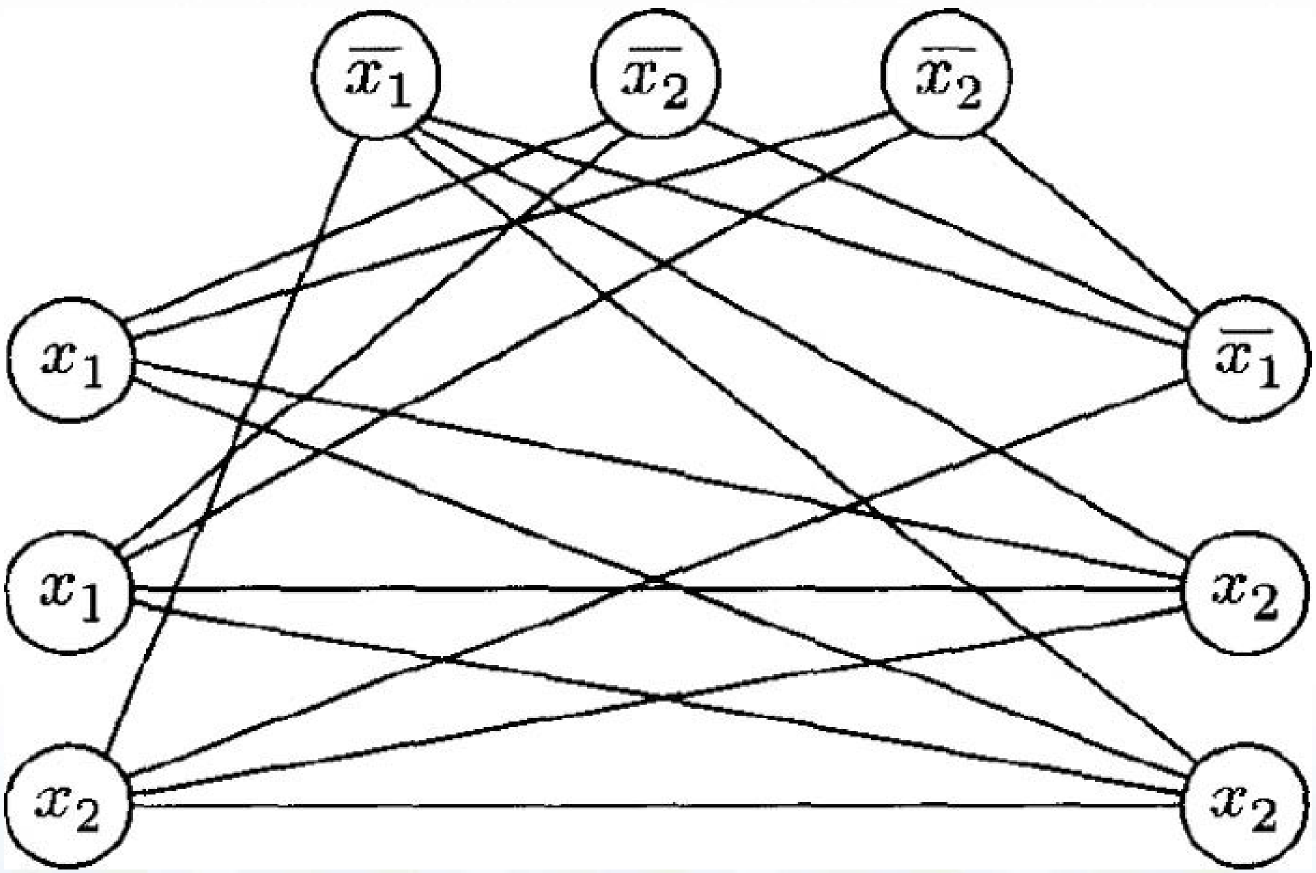
PROOF IDEA

Take any 3*SAT* formula and polynomial-time reduce it to a graph such that if the graph has a clique then the 3cnf-formula is satisfiable.

- Some details:
  - φ is a formula with k clauses each with 3 literals.
  - The k clauses in φ map to k groups of 3 nodes each called a triple.
  - Each node in the triple corresponds to one of the literals in the corresponding clause.
  - No edges between the nodes in a triple.
  - No edges between "conflicting" nodes (e.g., x and $\overline{x}$)

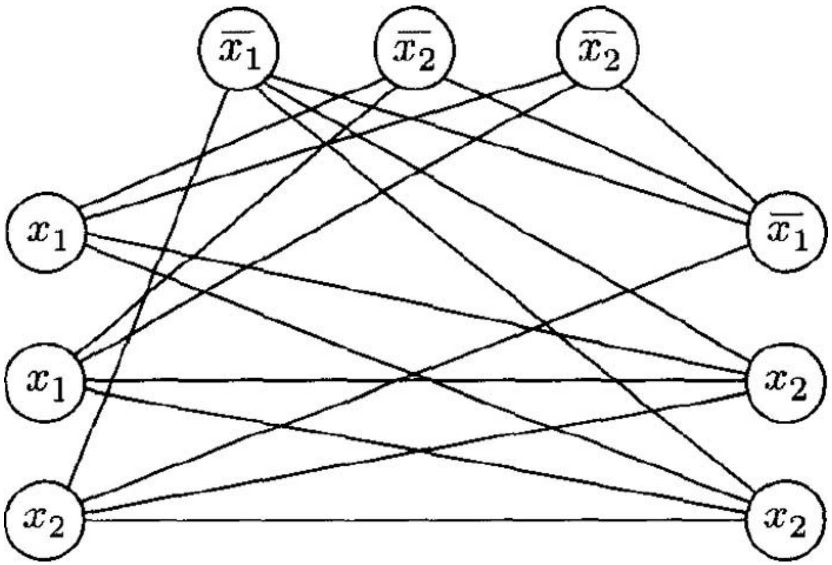
An Example Reduction: Reducing *SAT* to *CLIQUE*

$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



An Example Reduction: Reducing *SAT* to *CLIQUE*

$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



- If φ has a satisfying assignment, then at least one literal in each clause needs to be 1.
- We select the corresponding nodes in the corresponding triples.
- These nodes should form a k-clique.
- If G has a k-clique, then selected nodes give a satisfying assignment to variables.

NP - Completeness

DEFINITION – NP-COMPLETENESS

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

1. B is in NP, and
2. Every A in NP is polynomial time reducible to B.

THEOREM

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

THEOREM

If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

PROOF

All $A \leq_P B$ and $B \leq_P C$ thus all $A \leq_P C$.

The Cook – Levin Theorem

THEOREM

SAT is NP-Complete.

PROOF IDEA

- Showing *SAT* is in NP is easy.
  - Nondeterministically guess the assignments to variables and accept if the assignments satisfy φ
- We can encode the accepting computation history of a polynomial time NTM for every problem in NP as a *SAT* formula φ.
- Thus every language $A \in$ NP is polynomial-time reducible to *SAT*.
  - N is a NTM that can decide A in time $O(n^k)$
  - N accepts w if and only if φ is satisfiable.