Computing Theory

COMP 147 (4 units)

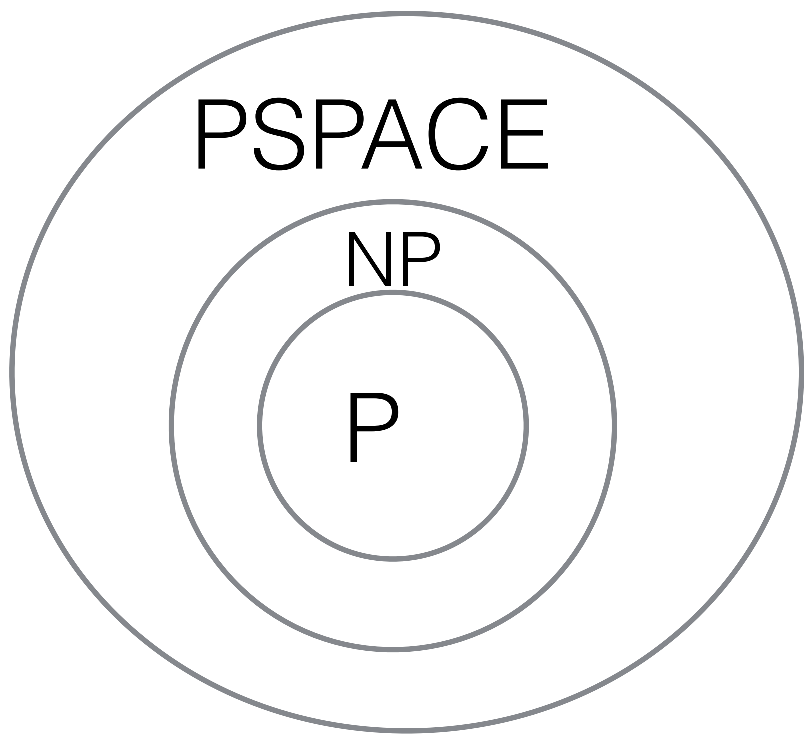
Chapter 7: Time Complexity
Class NP
NP Completeness

Last Time

- Section 7.1: Time Complexity
- Section 7.2: Class P

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

Complexity Classes



NP: nondeterministic polynomial time

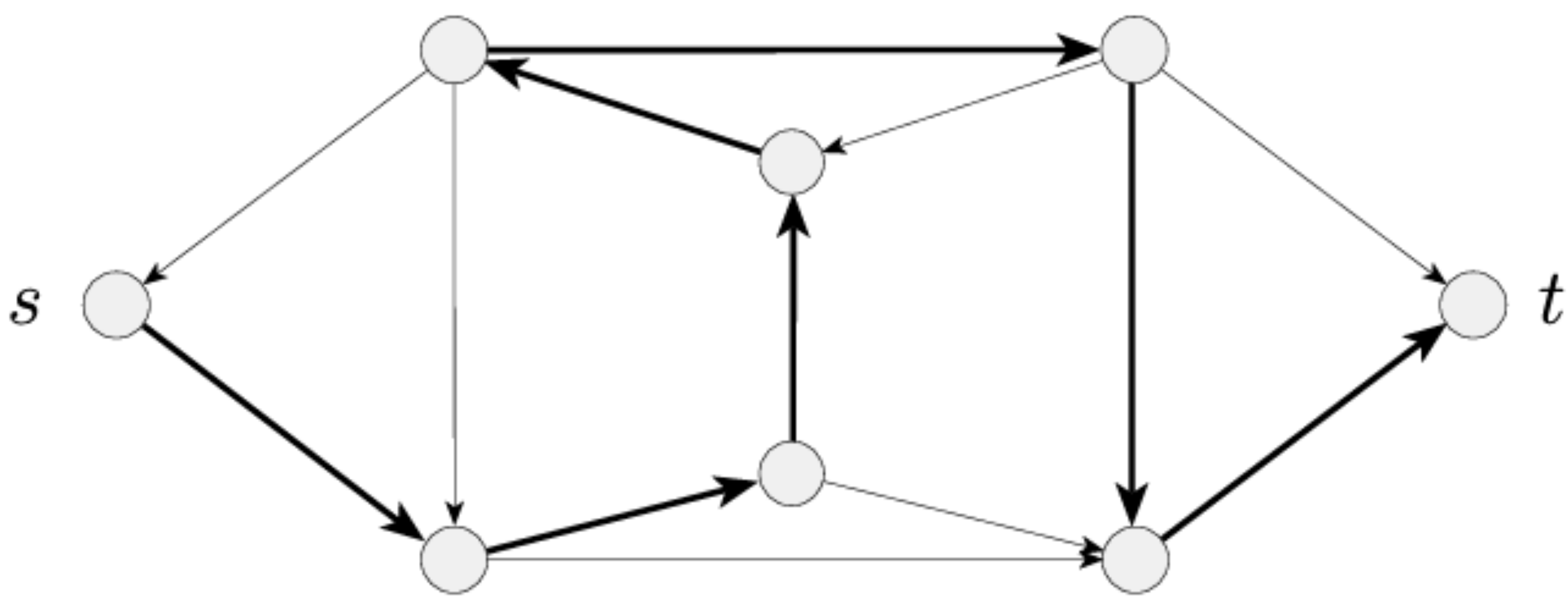
Decision problems

- Problem
  - Sort this list of numbers
  - Find the shortest path between s and t in a graph
- Decision Version
  - Is this list sorted?
  - Is there a shorter path less than k

$\boldsymbol{HAMPATH} \notin \mathrm{P}$?

- $HAMPATH$ = { <G,s,t> | G is a directed graph with a Hamiltonian path from s to t }

- A Hamiltonian path visits every node exactly once.



HAMPATH: Brute force algorithm

- On input $<G,s,t>$:
  - Generate a permutation of all nodes of G, call these sequence s_1, s_2, … , $s_{n!}$
  - For $i = 1$ to $n!$:
    - If s_{i} is a valid path from s to t, accept
  - Reject (all sequence failed test)

- This algorithm is $O(n!)$

- There is no known algorithm for HAMPATH ∈ P

Verifiers

DEFINITION 7.18

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

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

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.

- A verifier only needs to check if a solution is valid, it does not need to generate a solution

HAMPATH is polynomially verifiable

- $V_{HAMPATH}$ = { $<G,s,t>$, c | G is a directed graph and c is a Hamiltonian path from s to t }
  - Let $c = c_1, c_1, \ldots, c_n$
  - Verify that $c_1 = s$
  - Verify that $c_n = t$
  - For $i = 1$ to n-1
    - Verify that an edge exists from c_i to c_{i+1}
  - If all tests pass, accept, otherwise reject
- Complexity?

in polynomial time

Verifiers and Certificates

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

$$A = \{w | \ V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

- c is called a certificate
  - It certifies that a solution to the problem exists
  - Usually, c is simply a solution to the problem

The Class NP

- Definition 7.19:
  NP is the class of languages that have polynomial time verifiers.

- (True of False) If $A \in \text{P}$, then $A \in \text{NP}$

- $HAMPATH \in \text{NP}$
  - $HAMPATH \in \text{P}$?

P = The class of languages for which membership can be DECIDED quickly.*

NP = The class of languages for which membership can be VERIFIED quickly.

That is, given some information [the "certificate/proof"], you can quickly confirm that w is in the language.

NTIME

DEFINITION 7.21

$$\textbf{NTIME}(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

- Recall that the running time of an NTM is the length of the longest branch in the computation history

The Class NP

- Theorem 7.20
  A language is in NP iif it is decided by some nondeterministic polynomial time TM.

COROLLARY 7.22 ....

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

DEFINITION

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

THEOREM

A language is in NP iff it is decided by some NONDETERMINISTIC POLYNOMIAL-TIME Turing Machine

Sometimes this is given as the definition of "NP".

P = NP?

UNSOLVED QUESTION:

$P = NP$ } Which is it?
$P \subset NP$

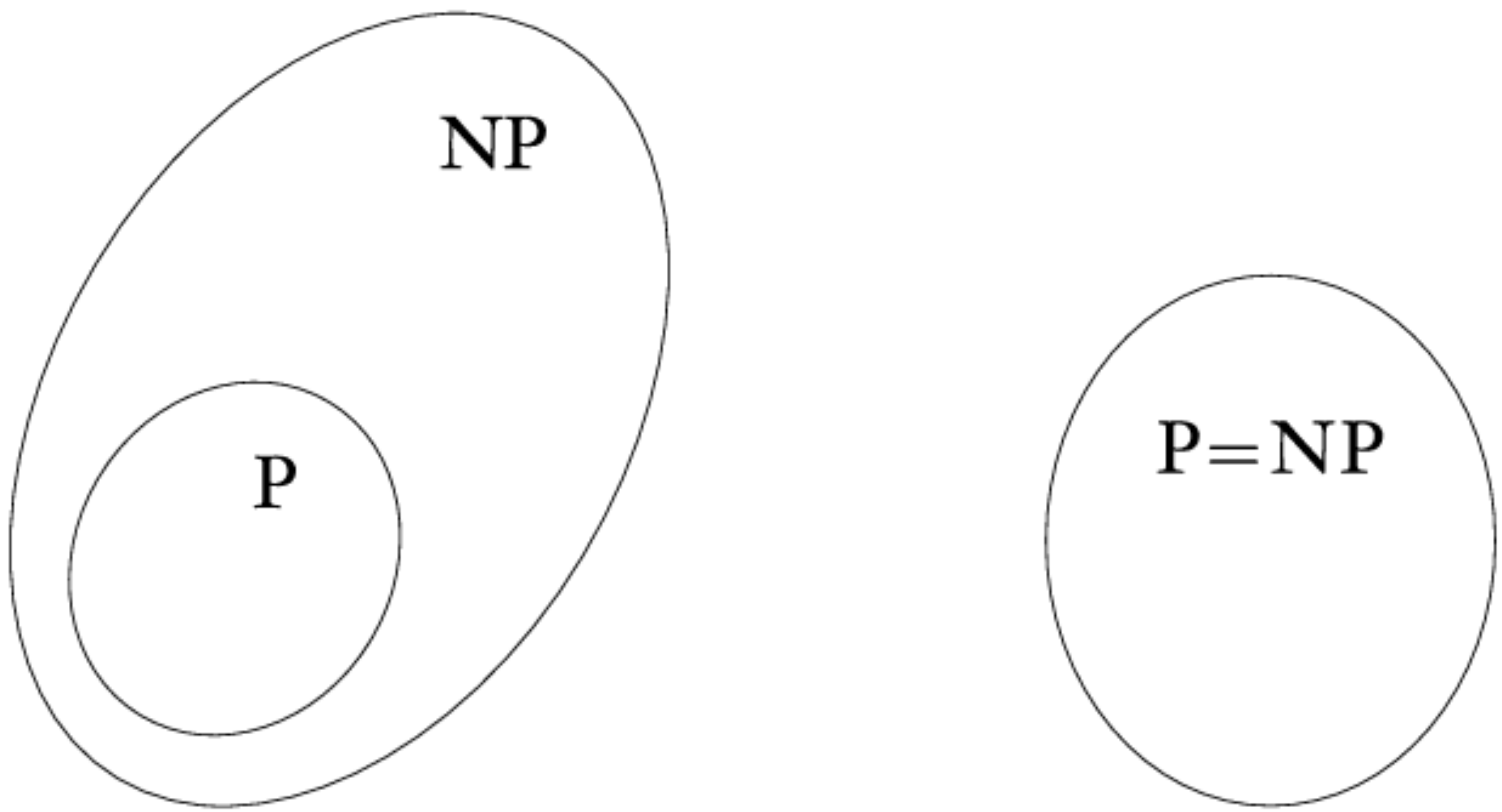


FIGURE 7.26
One of these two possibilities is correct

Best NP Bound (so far)

- The best deterministic method currently known for deciding languages in NP uses exponential time.

- We can prove that

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

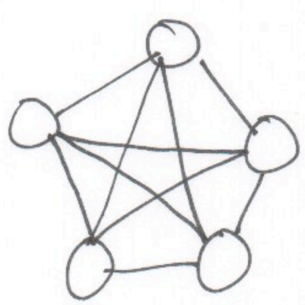
- We do not know if NP is contained in a smaller deterministic time complexity class.

The Clique Problem

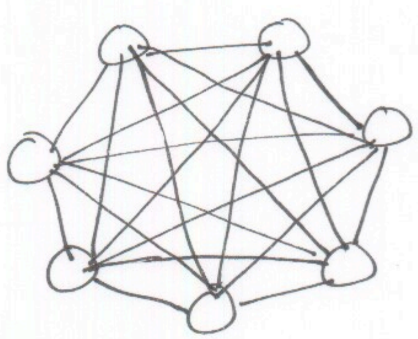
Given an undirected graph...

A "clique" is a set of nodes such that every node in the clique is connected to every other node in the clique.

A K-clique is a clique with K members.



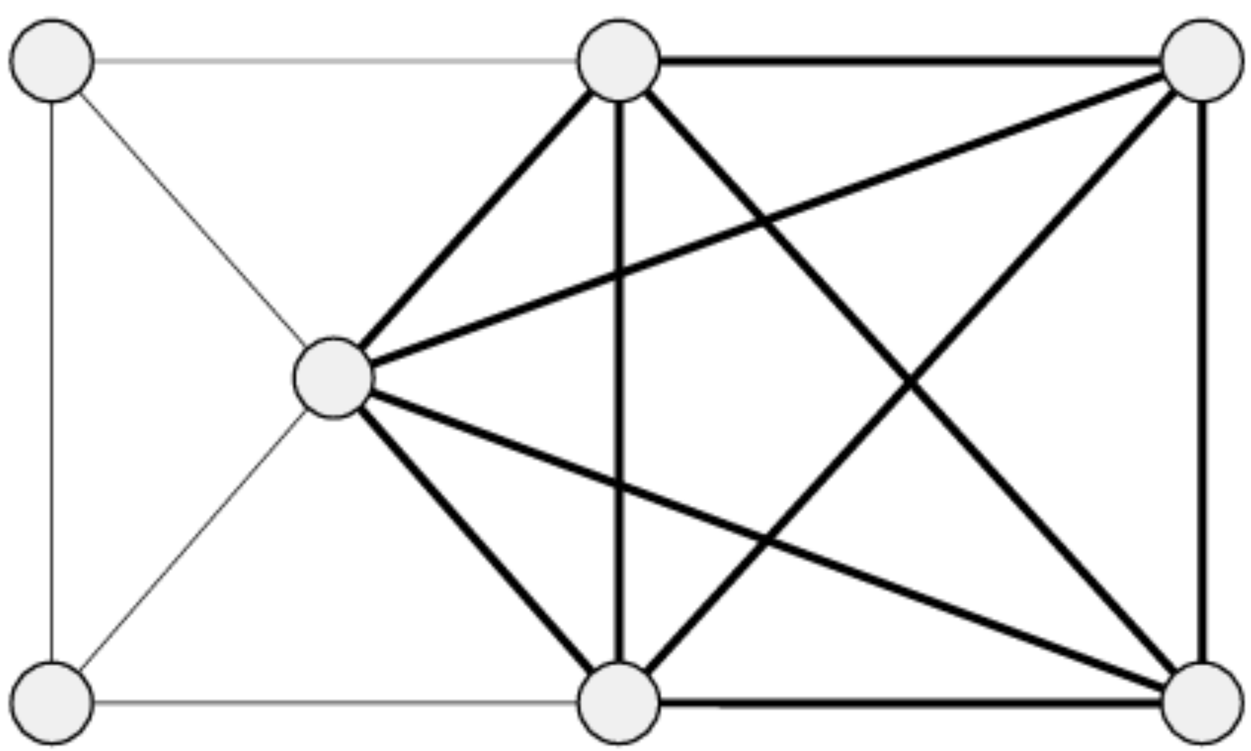
A 5-CLIQUE



A 7-CLIQUE.

CLIQUE

- $CLIQUE = \{ \langle G,k \rangle \mid G$ is an undirected graph with a k-clique $\}$
  - clique = fully connected subgraph
  - k-clique = fully connected subgraph of k nodes



Graph with a 5-clique

CLIQUE $\in$ NP

Using definition1: Polynomial time verifier

THEOREM 7.24

CLIQUE is in NP.

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

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

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

CLIQUE ∈ NP

Using definition2: Nondeterministic Turing Machines

THEOREM 7.24

CLIQUE is in NP.

ALTERNATIVE PROOF If you prefer to think of NP in terms of nondeterministic polynomial time Turing machines, you may prove this theorem by giving one that decides *CLIQUE*. Observe the similarity between the two proofs.

$N =$ "On input $\langle G, k\rangle$, where G is a graph:
1. Nondeterministically select a subset c of k nodes of G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, *accept*; otherwise, *reject*."

SAT $\in$ NP

Example: Determine the satisfiability of the following compound propositions:

$$(p \lor \neg q) \land (q \lor \neg r) \land (r \lor \neg p)$$

Solution: Satisfiable. Assign **T** to p, q, and r.

$$(p \lor q \lor r) \land (\neg p \lor \neg q \lor \neg r)$$

Solution: Satisfiable. Assign **T** to p and F to q.

$$(p \lor \neg q) \land (q \lor \neg r) \land (r \lor \neg p) \land (p \lor q \lor r) \land (\neg p \lor \neg q \lor \neg r)$$

Solution: Not satisfiable. Check each possible assignment of truth values to the propositional variables and none will make t
proposition true.

NP-Completeness

Complete Problems

- One way to address the **P** = **NP** question is to identify complete problems for NP.

- An NP-complete problem has the property that it is in **NP**, and if it is in **P**, then every problem in **NP** is also in **P**.

- Defined formally via “polytime reductions.”

NP-Completeness

- A problem L is NP-complete if
  - 1. $L \in$ NP, and
  - 2. Every problem L' $\in$ NP,
    L' is polytime reducible to L in polynomial time

- L is as hard as any problem in NP

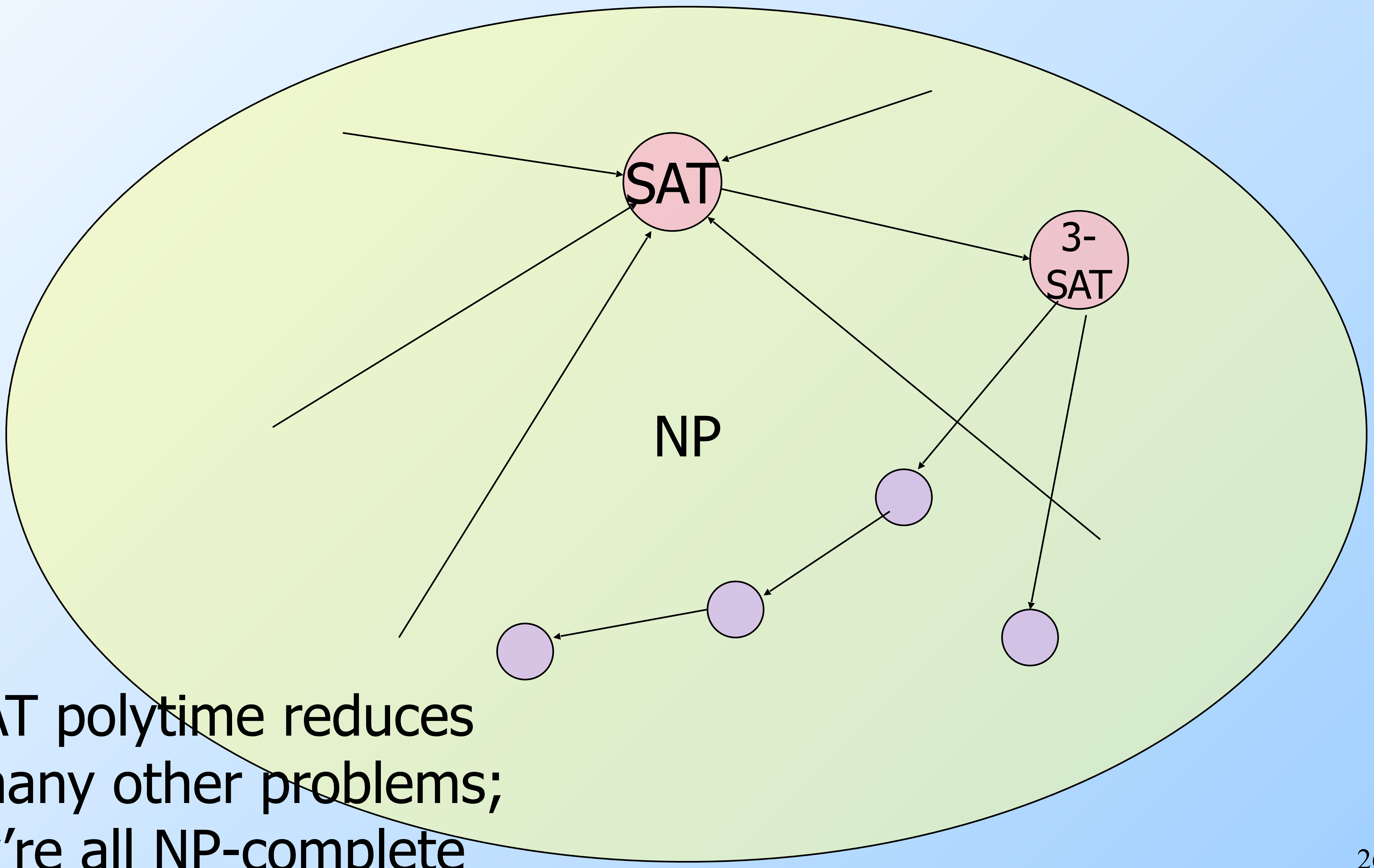
- If a polynomial time algorithm is *ever* found (on a deterministic machine) for any "NP-Complete" problem, then $P = NP$ follows!

∴ And polynomial time algorithms exist for all problems in NP!

The Plan

All of **NP** polytime reduces to SAT, which is therefore NP-complete

SAT polytime reduces to 3-SAT



3-SAT polytime reduces to many other problems; they're all NP-complete

Satisfiability ($\boldsymbol{SAT}$)

- SAT = { <ϕ> | ϕ is a satisfiable Boolean formula }

- Suppose the formula has m variables and n operations.
  - How many possible assignments? 2^m

- Given an assignment, what is the complexity of verification? $O(n)$

3SAT

- $3SAT$ = { <φ> | φ is a satisfiable 3cnf formula }

- literal: a Boolean variable or a negated Boolean variable
- clause: literals connected by disjunction (∨)
- cnf formula: clauses connected by conjunction (∧)
- 3cnf formula: each clause has exactly 3 literals

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

3SAT

- *3SAT* = { <ϕ> | ϕ is a satisfiable 3cnf formula }

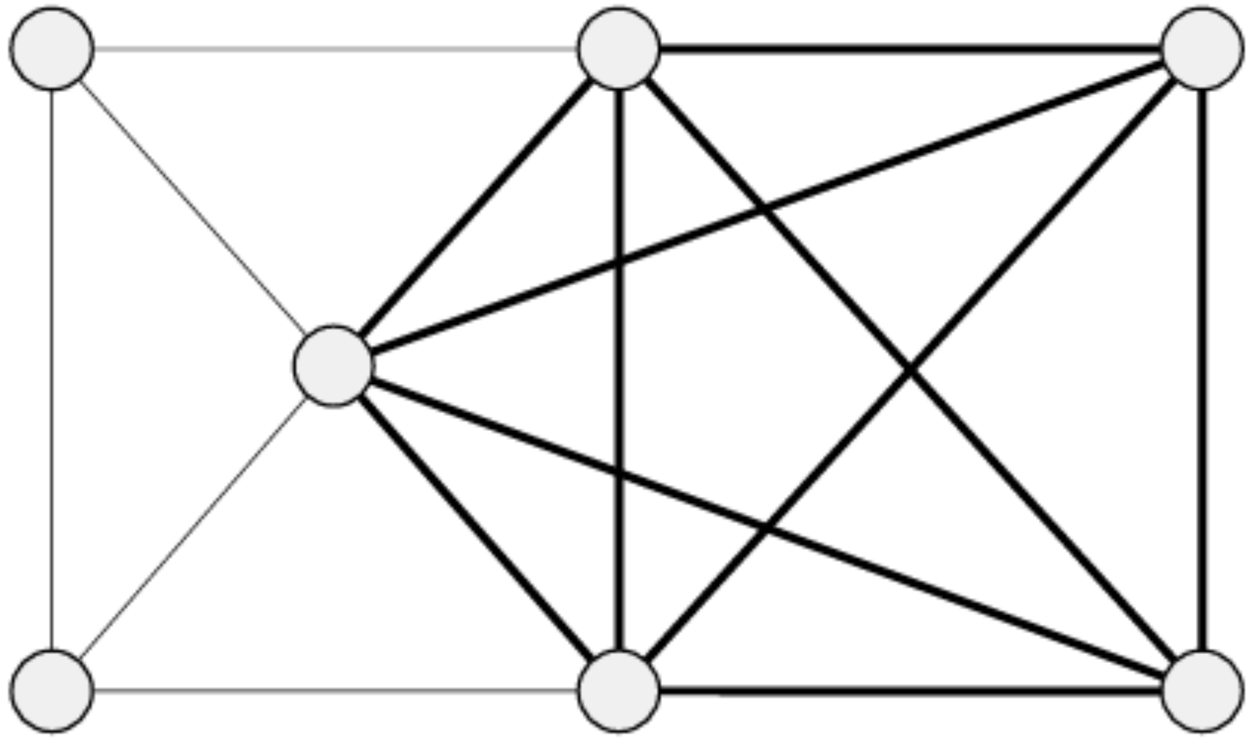
$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

- Each clause must have at least one true literal

- Example has 64 possible assignments

CLIQUE

- $CLIQUE = \{ \langle G,k \rangle \mid G$ is an undirected graph with a k-clique $\}$
  - clique = fully connected subgraph
  - k-clique = fully connected subgraph of k nodes

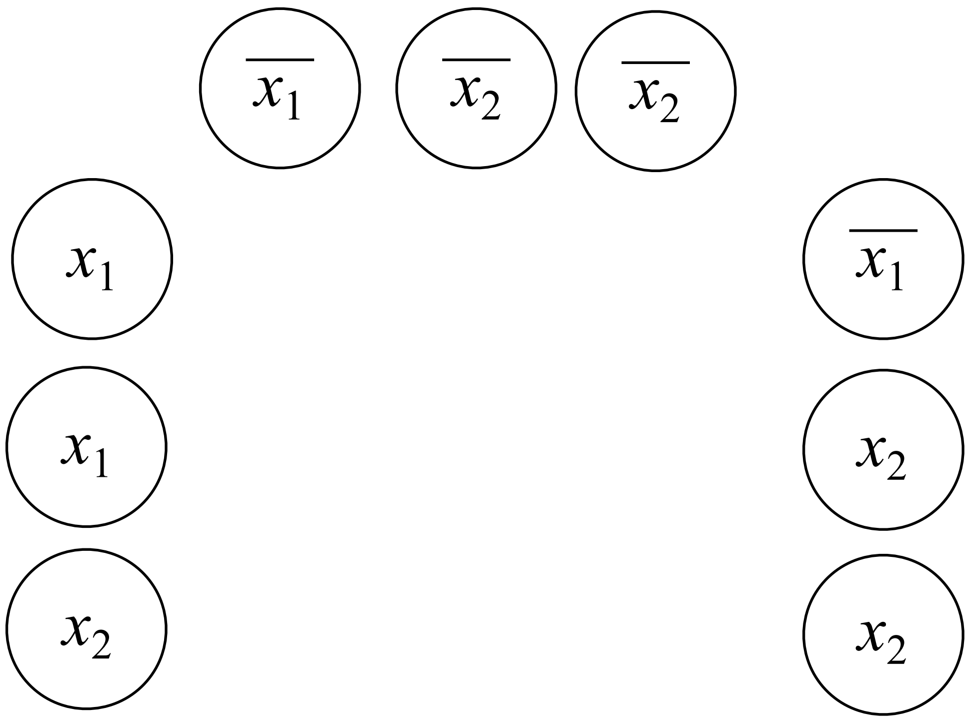


Graph with a 5-clique

3SAT $\leq_P$ *CLIQUE*

- Given a 3cnf-formula with k clauses, generate a graph with $3k$ nodes, such that the formula is satisfiable iff the graph has a k-clique.

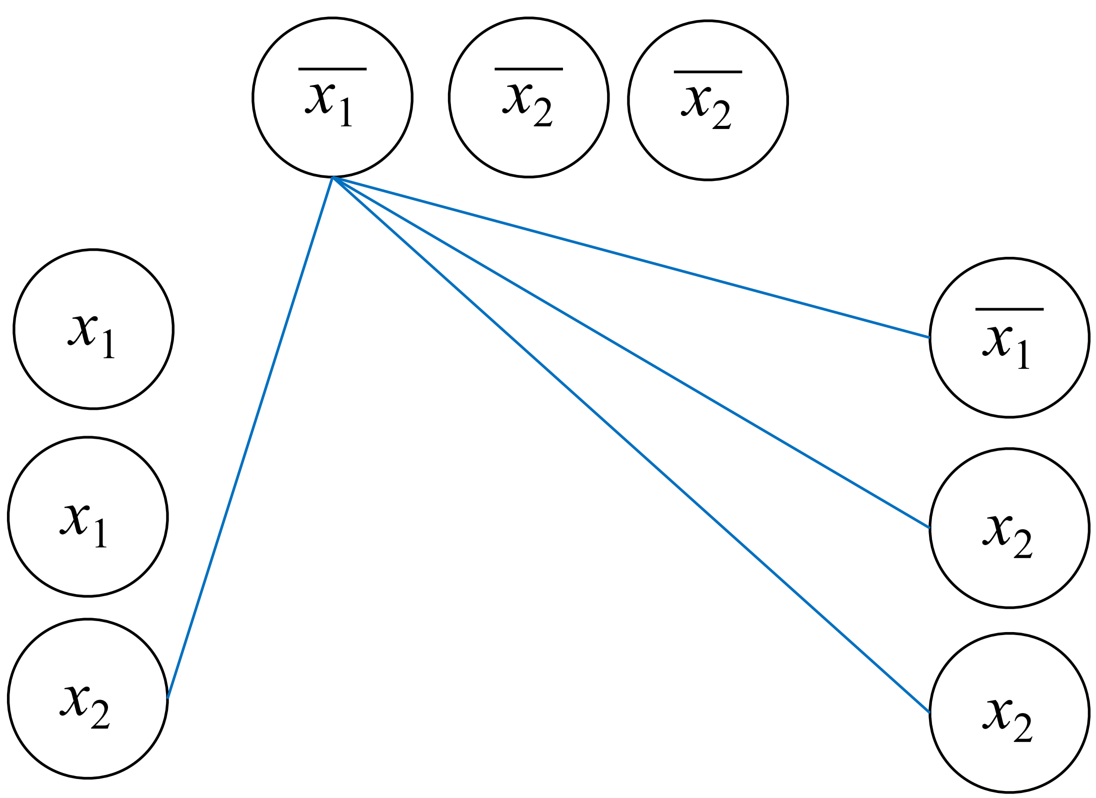
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Connect each pair of nodes, unless:
  - they are in the same clause, or
  - they have contradictory (negated) labels

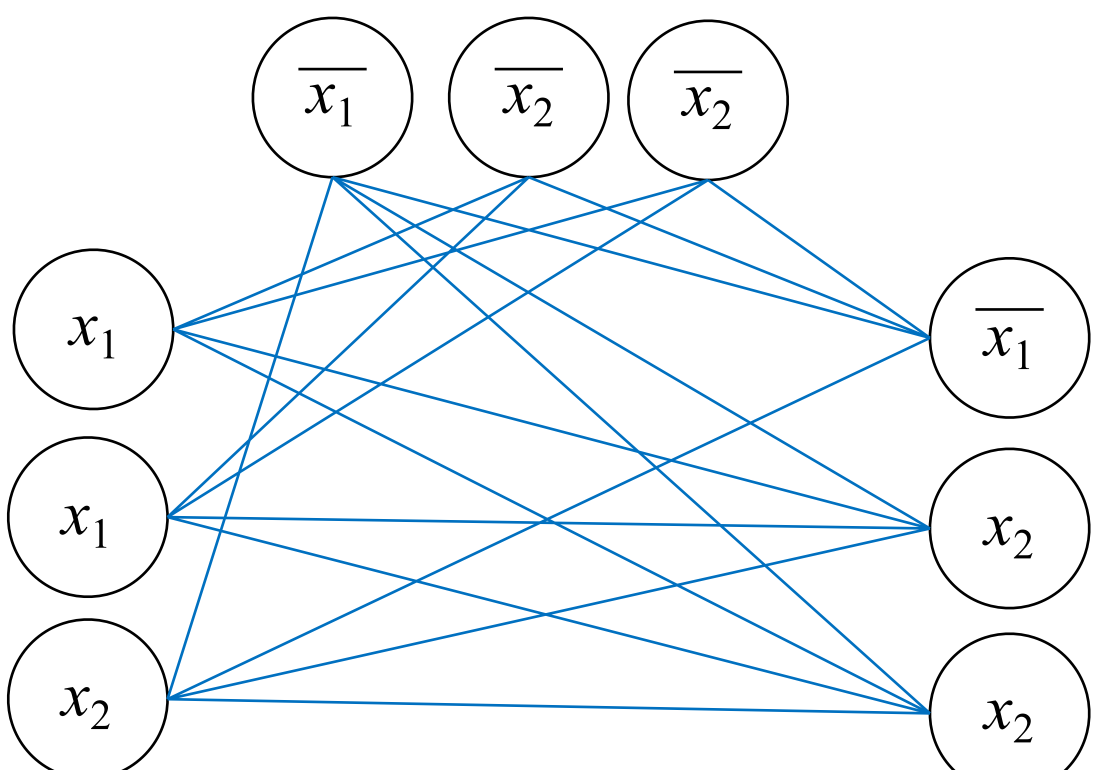
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Connect each pair of nodes, unless:
  - they are in the same clause, or
  - they have contradictory (negated) labels

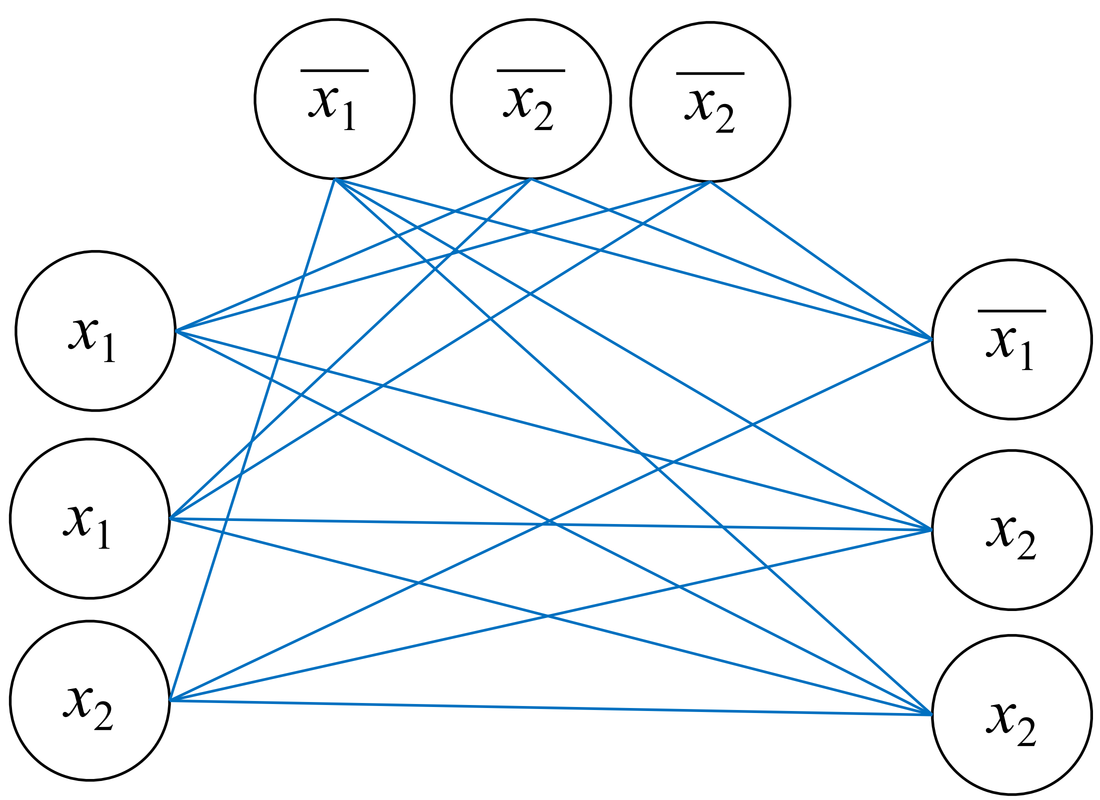
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Any k-clique
  - has at most one node from each clause → has exactly one node from each clause
  - Does not contain contradictory assignments

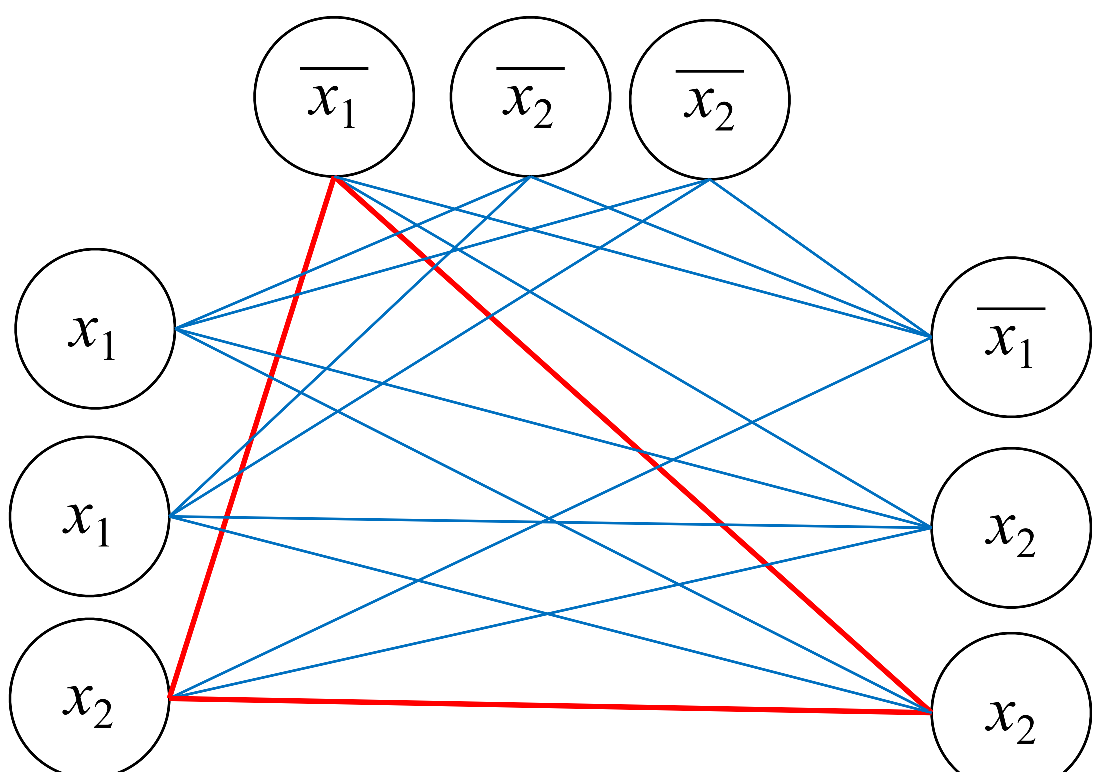
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Any k-clique
  - has at most one node from each clause → has exactly one node from each clause
  - Does not contain contradictory assignments

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



Initial NP-complete Language

- Theorem 7.37: $\boldsymbol{SAT}$ is NP-complete

- Proof Idea:
  - 1. Show that $SAT \in$ NP (easy)
  - 2. Show that <u>any</u> language $A \in$ NP
    is ptime-reducible to SAT

    Given <A,w> we'll construct a Boolean formula
    ϕ that simulates the NP machine M for A on w.
    M accepts $w \Leftrightarrow \phi$ is satisfiable
    M doesn't accept $w \Leftrightarrow \phi$ is not satisfiable

Given that AND, OR and NOT are the basic components of digital computers,
it's not surprising that we can simulate a TM with a logical formula.
However, the devil is in the details.

$\boldsymbol{SAT} \in$ NP

- A nondeterministic poly-time TM can guess an assignment for a given formula ϕ and accept if the assignment satisfies ϕ.

- Clearly, verification is $O(n^k)$, therefore SAT $\in$ NP