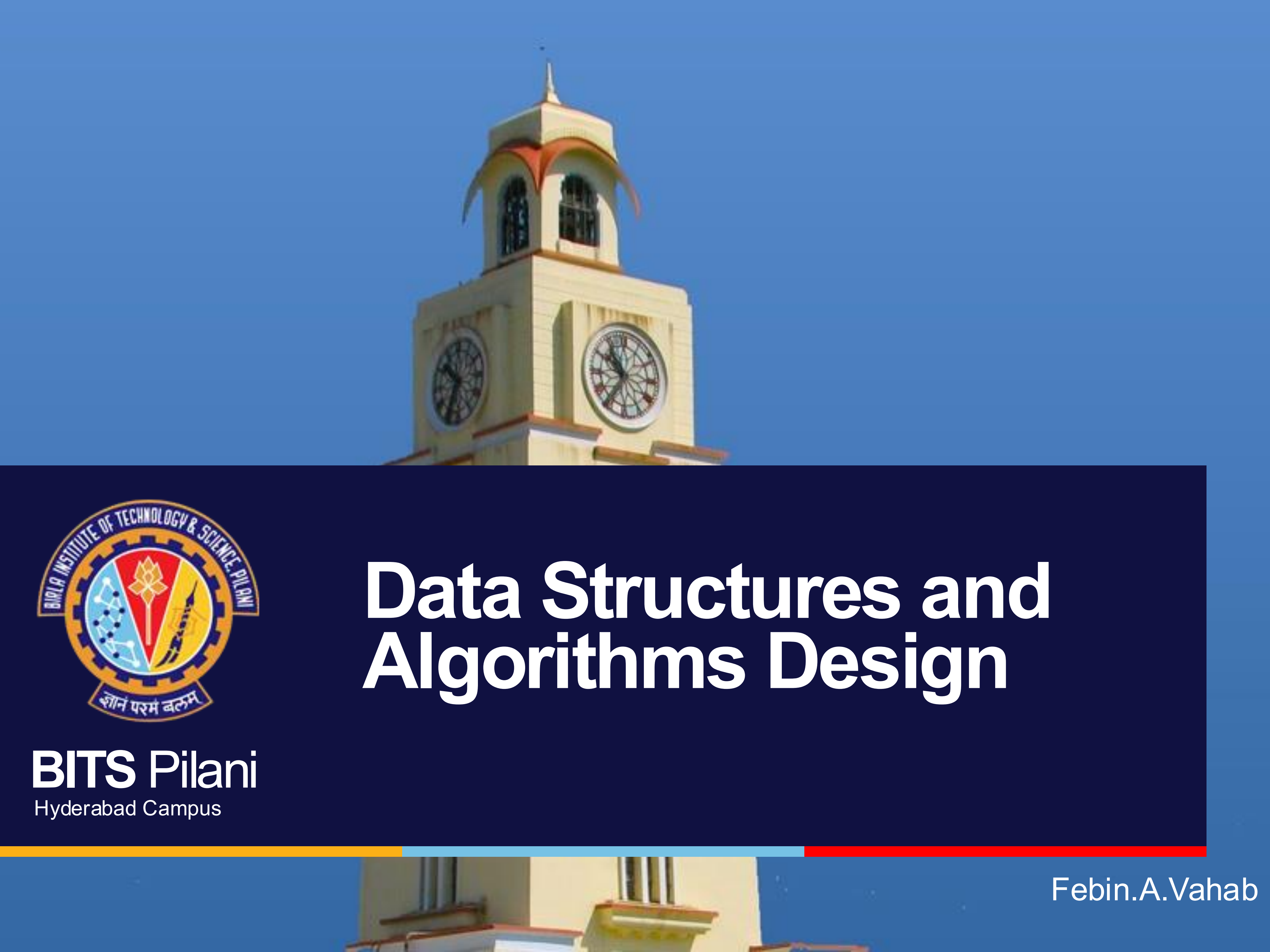


BITS Pilani
Hyderabad Campus

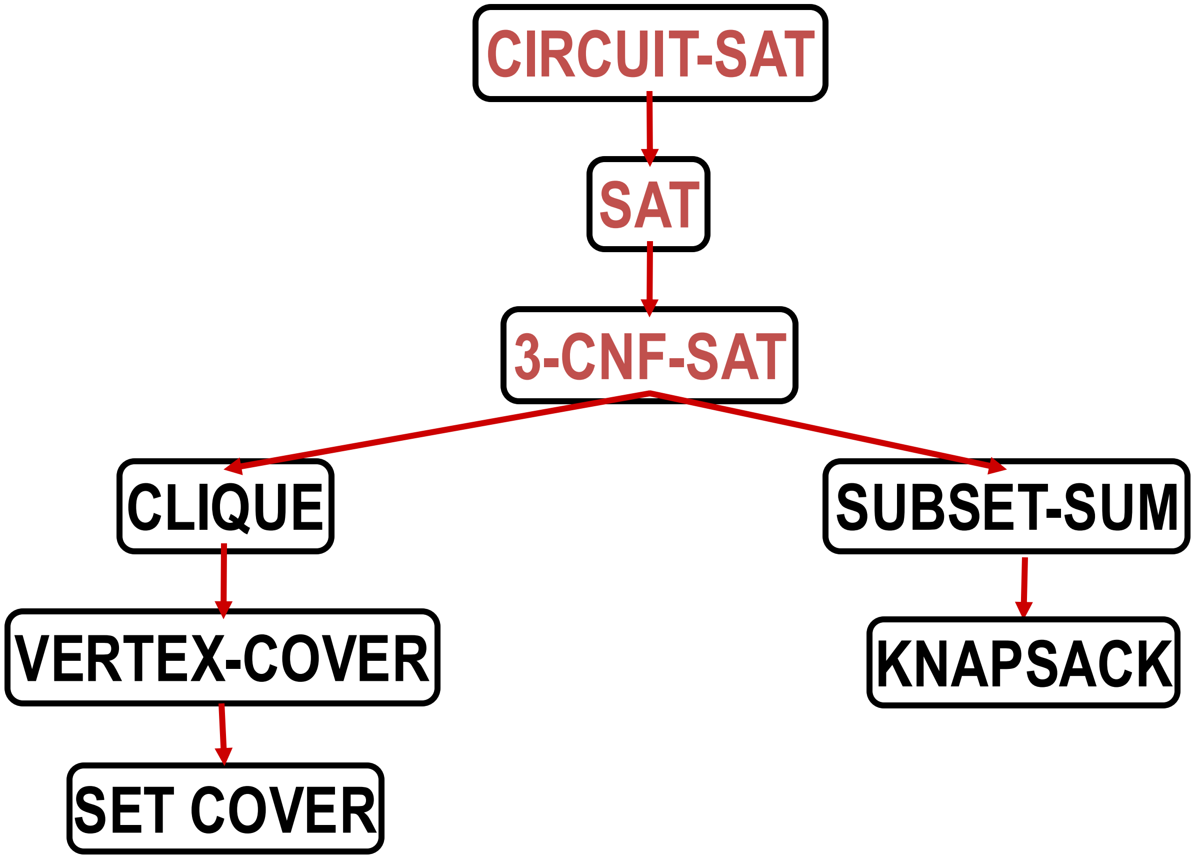
Data Structures and Algorithms Design

Febin.A.Vahab

CONTACT SESSION 14 -PLAN

Contact Sessions(#)	List of Topic Title	Text/Ref Book/external resource
14	Definition of P and NP classes and examples, Understanding NP-Completeness: CNF-SAT Cook-Levin theorem Polynomial time reducibility: CNF-SAT and 3-SAT, Vertex Cover	T1: 13.1, 13.2, 13.3
15	Polynomial time reducibility: Clique and Vertex-Cover	T1: 13.3, 13.4

NP-Completeness and the Proofs

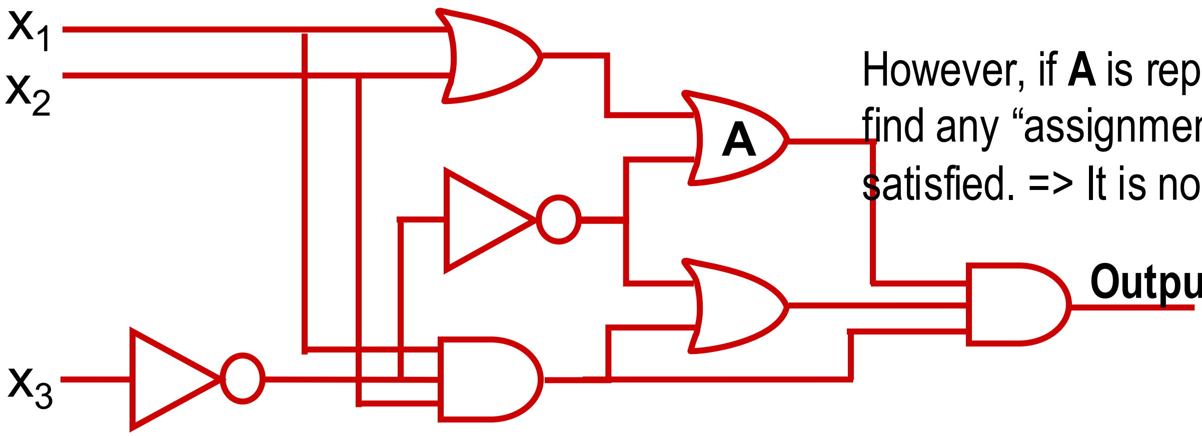


CIRCUIT-SAT
First NP-complete Problem

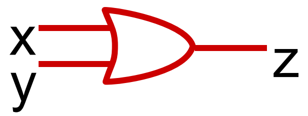
- Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

If we assign 1, 1, 0 to x_1, x_2, and x_3, the output will be 1. The circuit is satisfied by some assignment .It is satisfiable.

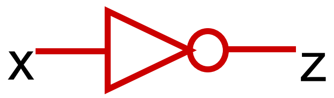
However, if **A** is replaced by an AND gate, we can't find any "assignment" of (x_1, x_2, x_3) to make it satisfied. => It is not satisfiable



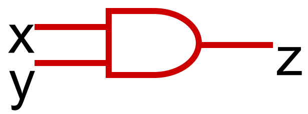
CIRCUIT-SAT First NP-complete Problem



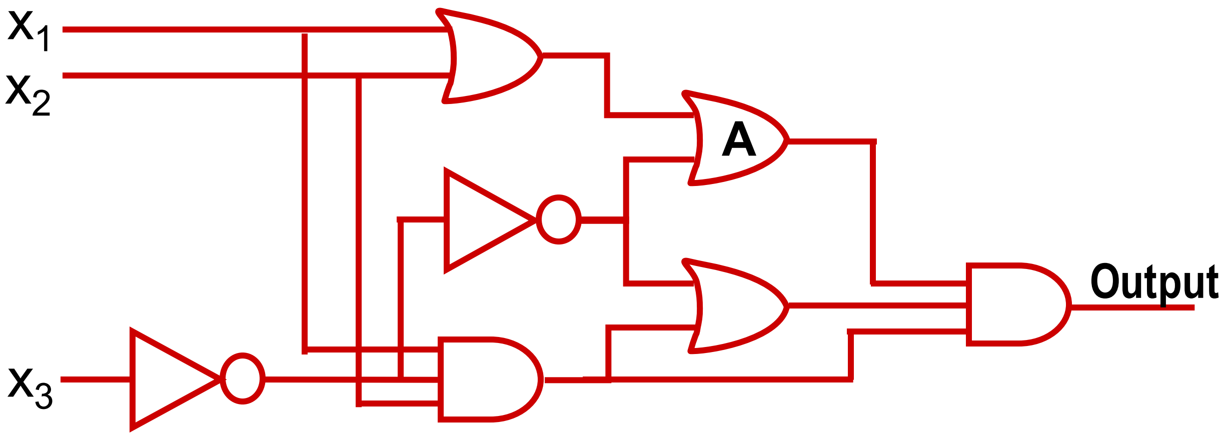
x	y	x∨y
0	0	0
0	1	1
1	0	1
1	1	1



x	¬x
0	1
1	0



x	y	x∧y
0	0	0
0	1	0
1	0	0
1	1	1



NP-Completeness and the Proofs

- **A common approach to prove that a problem, A, is NP-complete:**

1. Prove **A** $\in$ NP
2. Select a known NP-complete problem **K**
3. Describe an algorithm that maps an instance of **K** to an instance of **A**
4. Prove that the results for both instances (yes or no) in (3) are the same
5. Prove that the algorithm in (3) runs in polynomial time [Polynomial-time Reducibility]

SAT Problem

- Given a boolean formula of n boolean variables, m boolean connectives, and required parenthesis, is it satisfiable?
- Boolean functions:

P	Q	$P \wedge Q$	$P \vee Q$	$\neg P$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

SAT Problem

Example: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

It is satisfiable by the assignment:

$x_1=0, x_2=0, x_3=1, x_4=1$

ie $((0 \rightarrow 0) \vee \neg ((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0$

$= (1 \vee \neg(1 \vee 1)) \wedge 1$

P	Q	$P \wedge Q$	$P \vee Q$	$\neg P$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

SAT Problem

- **SAT is NP-Complete**
- 2 parts of the proof:
  - **SAT** $\in$NP
  - **SAT** is NP-hard

– **<u>SAT$\in$NP</u>**

- Consider an algorithm:

bool Verify_Sat(Input_Boolean_Formula,
Certificate/*an assignment to the variables, eg. x_1, x_2, x_3*/)

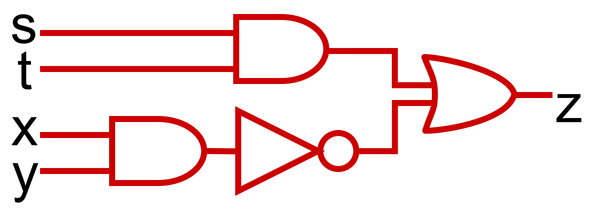
- This algorithm replaces each variable in the formula with its corresponding values and evaluate the expression.
- This is a 2-input, polynomial-time verification algorithm for SAT. Since we can find such an algorithm for **SAT**, we say that SAT can be verified in polynomial time.

SAT Problem

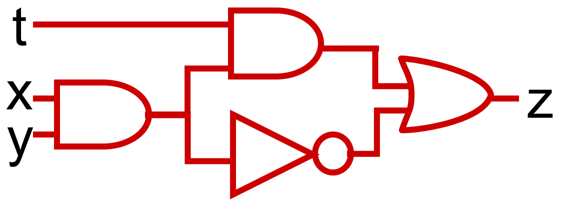
SAT is NP-hard

Show that CIRCUIT-SAT $\leq_p$ SAT

ie. Any instance of circuit satisfiability can be reduced in polynomial time to an instance of formula satisfiability



$z = (s \wedge t) \vee (\neg(x \wedge y))$



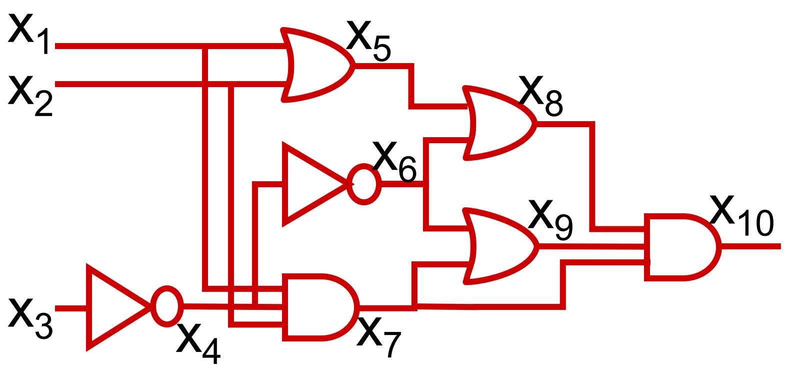
$z = (t \wedge (x \wedge y)) \vee (\neg(x \wedge y))$

SAT Problem

- Since we need to show that "Any instance of circuit satisfiability can be reduced in **polynomial time** to an instance of formula satisfiability."
- We design a more clever method:
- Step 1. For each gate, formulate it with the operation on its incident wires.

  eg. $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$
- Step 2. "AND" all the formulas of the gates.

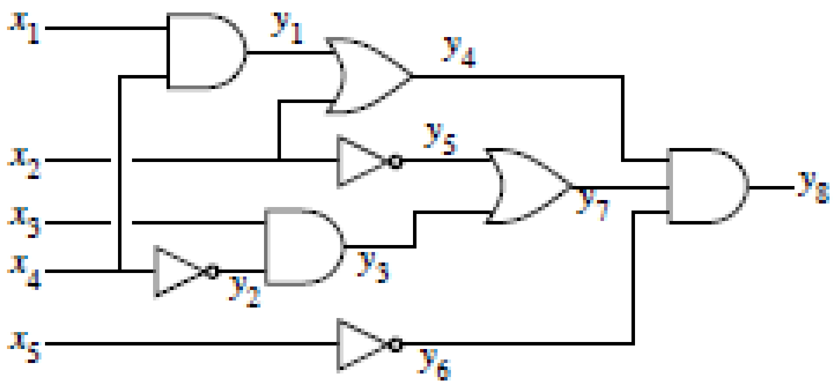
SAT Problem



The formula:

$$
\begin{aligned}
x_{10} \land\ & (x_4 \leftrightarrow \neg x_3) \\
\land\ & (x_5 \leftrightarrow (x_1 \lor x_2)) \\
\land\ & (x_6 \leftrightarrow \neg x_4) \\
\land\ & (x_7 \leftrightarrow (x_1 \land x_2 \land x_4)) \\
\land\ & (x_8 \leftrightarrow (x_5 \lor x_6)) \\
\land\ & (x_9 \leftrightarrow (x_6 \lor x_7)) \\
\land\ & (x_{10} \leftrightarrow (x_7 \land x_8 \land x_9))
\end{aligned}
$$

SAT Problem-Example



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (y_8 = y_4 \wedge y_7 \wedge y_6) \wedge y_8$$

A boolean circuit with gate variables added, and an equivalent boolean formula.

SAT –Reduction Picture

$$\begin{tikzcd}
\boxed{\text{boolean circuit}} \arrow[r, "O(n)"] & \boxed{\text{boolean formula}} \arrow[d, Rightarrow, "\text{SAT}"] \\
 & \boxed{\text{True or False}}
\end{tikzcd}$$

SAT Problem

- The original circuit is satisfiable iff the resulting formula is satisfiable
- We can transform any boolean circuit into a formula in linear time and the size of the resulting formula is only a constant factor larger than the size of the circuit
- Thus we’ve shown that if we had a polynomial-time algorithm for SAT, then we’d have a polynomial-time algorithm for Circuit Satisfiability
- This means that SAT is NP-Hard

The 3-CNF-SAT Problem

- Given a boolean formula in 3-CNF, is it satisfiable?
- **3-CNF-SAT is NP-complete**
- 2 parts of the proof:
  - A. 3-CNF-SAT $\in$NP
  - B. 3-CNF-SAT is NP-hard

A 3-CNF Example:

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

The 3-CNF-SAT Problem

3-CNF-SAT∈NP

Consider an algorithm:

```
bool Verify_3_CNF_SAT(Input_Boolean_Formula,
        Certificate /*an assignment to the variables*/ )
```

- This algorithm replaces each variable in the formula with its corresponding values and evaluate the expression.
- This is a 2-input, polynomial-time verification algorithm for 3-CNF-SAT.

The 3-CNF-SAT Problem

- Since we can find such an algorithm for 3-CNF-**SAT**, we say that 3-CNF-SAT can be verified in polynomial time, and 3-CNF-SAT$\in$NP.

The 3-CNF-SAT Problem

3-CNF-SAT is NP-hard

Show that SAT $\leq_p$ 3-CNF-SAT

ie. Any instance of formula satisfiability can be reduced in polynomial time to an instance of 3-CNF formula satisfiability.

A formula Example:

$((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

A 3-CNF Example:

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

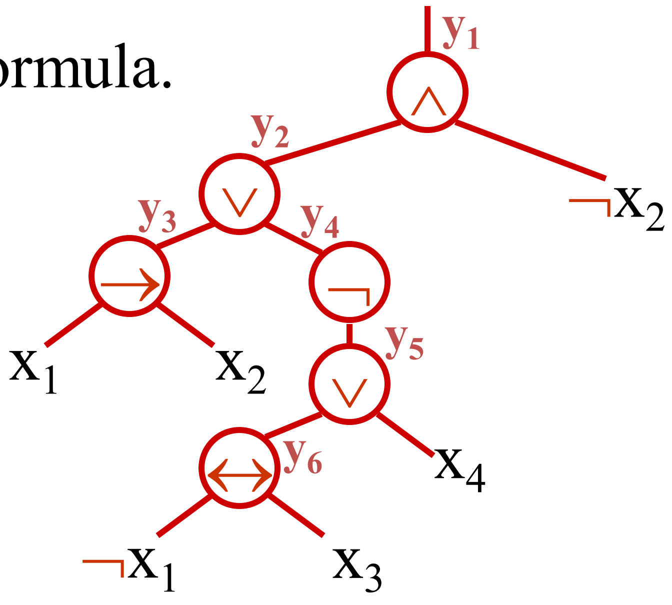
The 3-CNF-SAT Problem

The Reduction :

Step 1: Create a binary "parse" tree for the formula.

Step 2: Rewrite it in the form:

$$
\begin{aligned}
y_1 \;&\wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2))\\
&\wedge (y_2 \leftrightarrow (y_3 \vee y_4))\\
&\wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2))\\
&\wedge (y_4 \leftrightarrow \neg y_5)\\
&\wedge (y_5 \leftrightarrow (y_6 \vee x_4))\\
&\wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\end{aligned}
$$



The 3-CNF-SAT Problem

The Reduction :

Step 3: Change each sub-clause of the following to an OR of literals:

$$\begin{aligned} & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_4 \leftrightarrow \neg y_5) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \vee x_3)) \end{aligned}$$

Eg.

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

$(y_1 \wedge y_2 \wedge x_2)$

$(y_1 \wedge \neg y_2 \wedge x_2)$

$(y_1 \wedge \neg y_2 \wedge \neg x_2)$

$(\neg y_1 \wedge y_2 \wedge \neg x_2)$

We can rewrite $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ as:

$$\neg \left[(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2) \right]$$

By DeMorgan's laws:

$\Rightarrow \neg(y_1 \wedge y_2 \wedge x_2) \wedge \neg(y_1 \wedge \neg y_2 \wedge x_2) \wedge \neg(y_1 \wedge \neg y_2 \wedge \neg x_2) \wedge \neg(\neg y_1 \wedge y_2 \wedge \neg x_2)$

$\Rightarrow (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$

The 3-CNF-SAT Problem

The Reduction :

Step 3: Change each sub-clause to an OR of literals:

$$
\begin{aligned}
y_1 \wedge & (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
\wedge & (y_2 \leftrightarrow (y_3 \vee y_4)) \\
\wedge & (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
\wedge & (y_4 \leftrightarrow \neg y_5) \\
\wedge & (y_5 \leftrightarrow (y_6 \vee x_4)) \\
\wedge & (y_6 \leftrightarrow (\neg x_1 \vee x_3))
\end{aligned}
$$

We can rewrite $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ as:

$$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

Apply the same method to other sub-clauses:

$$
\begin{aligned}
y_1 \wedge & (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2) \\
\wedge & (y_2 \leftrightarrow (y_3 \vee y_4)) \\
\wedge & (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
\wedge & (y_4 \leftrightarrow \neg y_5) \\
\wedge & (y_5 \leftrightarrow (y_6 \vee x_4)) \\
\wedge & (y_6 \leftrightarrow (\neg x_1 \vee x_3))
\end{aligned}
$$

The 3-CNF-SAT Problem

- In 3SAT every clause must have exactly 3 different literals.
- To reduce from an instance of SAT to an instance of 3SAT, we must make all clauses to have exactly 3 variables...
- Basic idea
- (A) Pad short clauses so they have 3 literals.
- (B) Break long clauses into shorter clauses.
- (C) Repeat the above till we have a 3CNF.

The 3-CNF-SAT Problem

(A) *Case clause with one literal:* Let c be a clause with a single literal (i.e., $c = \ell$). Let u, v be new variables. Consider

$$\begin{aligned} c' = \ & \left(\ell \vee u \vee v\right) \wedge \left(\ell \vee u \vee \neg v\right) \\ & \wedge \left(\ell \vee \neg u \vee v\right) \wedge \left(\ell \vee \neg u \vee \neg v\right). \end{aligned}$$

Observe that c' is satisfiable iff c is satisfiable

Case clause with 2 literals: Let $c = \ell_1 \vee \ell_2$. Let u be a new variable. Consider

$$c' = \left(\ell_1 \vee \ell_2 \vee u\right) \wedge \left(\ell_1 \vee \ell_2 \vee \neg u\right).$$

Again c is satisfiable iff c' is satisfiable

The 3-CNF-SAT Problem

Clauses with more than 3 literals

Let $c = \ell_1 \vee \cdots \vee \ell_k$. Let $u_1, \ldots u_{k-3}$ be new variables. Consider

$$c' = \left(\ell_1 \vee \ell_2 \vee u_1\right) \wedge \left(\ell_3 \vee \neg u_1 \vee u_2\right) \wedge \left(\ell_4 \vee \neg u_2 \vee u_3\right) \wedge \cdots \wedge \left(\ell_{k-2} \vee \neg u_{k-4} \vee u_{k-3}\right) \wedge \left(\ell_{k-1} \vee \ell_k \vee \neg u_{k-3}\right).$$

The clause with more than 3 variables {a1,a2,a3,a4,a5} can be expanded to {a1,a2,s1}{!s1,a3,s2}{!s2,a4,a5} with s1 and s2 new variables whose value will depend on which variable in the original clause is true

The 3-CNF-SAT Problem

Example .

$$\begin{aligned}\varphi = &\Big(\neg x_1 \vee \neg x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \\ &\wedge\Big(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\Big) \wedge \Big(x_1\Big) .\end{aligned}$$

Equivalent form:

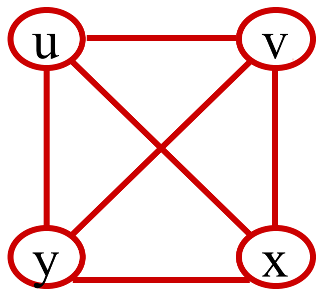
$$\begin{aligned}\psi = &(\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z) \\ &\wedge (x_1 \vee \neg x_2 \vee \neg x_3) \\ &\wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1) \\ &\wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v) \\ &\wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v) .\end{aligned}$$

The 3-CNF-SAT Problem

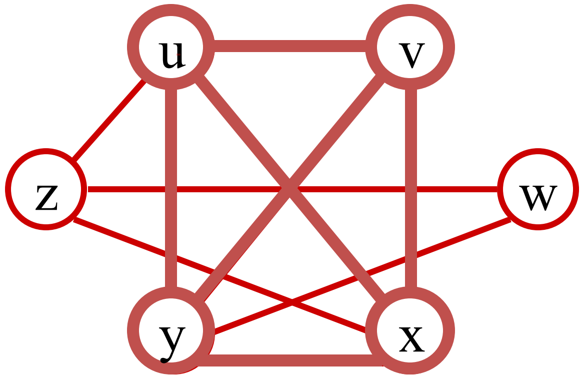
```
ReduceSATTo3SAT($\varphi$):
    // $\varphi$: CNF formula.
    for each clause $c$ of $\varphi$ do
        if $c$ does not have exactly 3 literals then
            construct $c'$ as before
        else
            $c' = c$
    $\psi$ is conjunction of all $c'$ constructed in loop
    return Solver3SAT($\psi$)
```

The Clique Problem

A **complete** graph



A **Clique** in an undirected graph G=(V,E) is a complete subgraph of G.



$\boxed{4}$

$\boxed{4}$

Optimization problem: Find a clique of maximum size in a graph. ✓

Decision problem: Whether a clique of a given size k exists in the graph. ?

4

The Clique Problem

- A ***clique*** in an undirected graph G (V,E) is a subset V’ *subset of* V, of vertices, each pair of which is connected by an edge in E.
- In other words, a clique is a complete subgraph of G.
- The ***size*** of a clique is the number of vertices it contains.

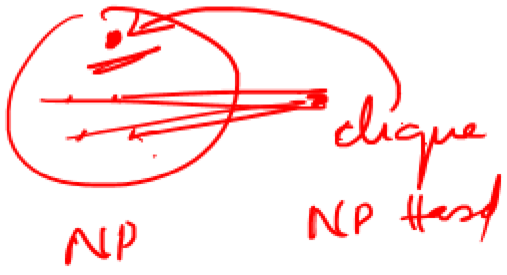
The CLIQUE Problem

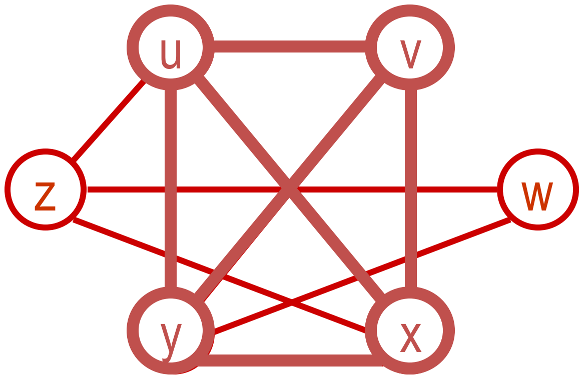
CLIQUE is NP-complete ✓

2 parts of the proof:

A. CLIQUE ∈NP ✓

B. CLIQUE is NP-hard ✓





The CLIQUE Problem

- **CLIQUE $\in$NP** ✓

$\{u, v, x, y\}$ choice( )

- Consider an algorithm:

bool Verify_CLIQUE(Input_Graph, Certificate /*a set of vertices in the input graph*/)

- This algorithm checks whether the set of vertices in the certificate are linked up as a complete graph.
- This is a 2-input, polynomial-time verification algorithm for CLIQUE.
- Since we can find such an algorithm for CLIQUE, we say that CLIQUE can be verified in polynomial time, and CLIQUE$\in$NP.

The CLIQUE Problem

- **CLIQUE is NP-hard** ✓
- Show that 3-CNF-SAT $\leq_p$ CLIQUE ✓
- ie. Any instance of 3-CNF formula satisfiability can be reduced in polynomial time to an instance of CLIQUE.

A 3-CNF Example:
$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ ✓

The CLIQUE Problem

- **The Reduction :**
- Describe a 3-CNF formula with k sub-clauses as:
- $(l^1_1 \vee l^1_2 \vee l^1_3) \wedge (l^2_1 \vee l^2_2 \vee l^2_3) \wedge .. (l^k_1 \vee l^k_2 \vee l^k_3)$
- **Reduce it to a clique problem such that it is satisfiable if and only if a corresponding graph has a clique of size k.**
- Step 1: Represent each “term” l^r_i as a vertex v^r_i.
- Step 2: Create the edges for any two vertices: v^r_i and v^s_j if: $\underline{r \neq s}$ and $\underline{l^r_i \text{ is not the negation of } l^s_j}$

The CLIQUE Problem

- **The Reduction :**
- The reduction algorithm begins with an instance of 3-CNF-SAT.
- Let Ø =C1 ^ C2 ^…^ Ck be a boolean formula in 3-CNF with k clauses.
- For r =1;2;….k, each clause Cr has exactly three distinct literals $\boldsymbol{l^r_1}$, $\boldsymbol{l^r_2}$, ***and*** $\boldsymbol{l^r_3}$.
- We shall construct a graph G such that is satisfiable if and only if G has a clique of size k.

The CLIQUE Problem

We construct the graph $G = (V, E)$ as follows. For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, we place a triple of vertices v_1^r, v_2^r, and v_3^r into V. We put an edge between two vertices v_i^r and v_j^s if both of the following hold:

- v_i^r and v_j^s are in different triples, that is, $r \neq s$, and
- their corresponding literals are *consistent*, that is, l_i^r is not the negation of l_j^s.

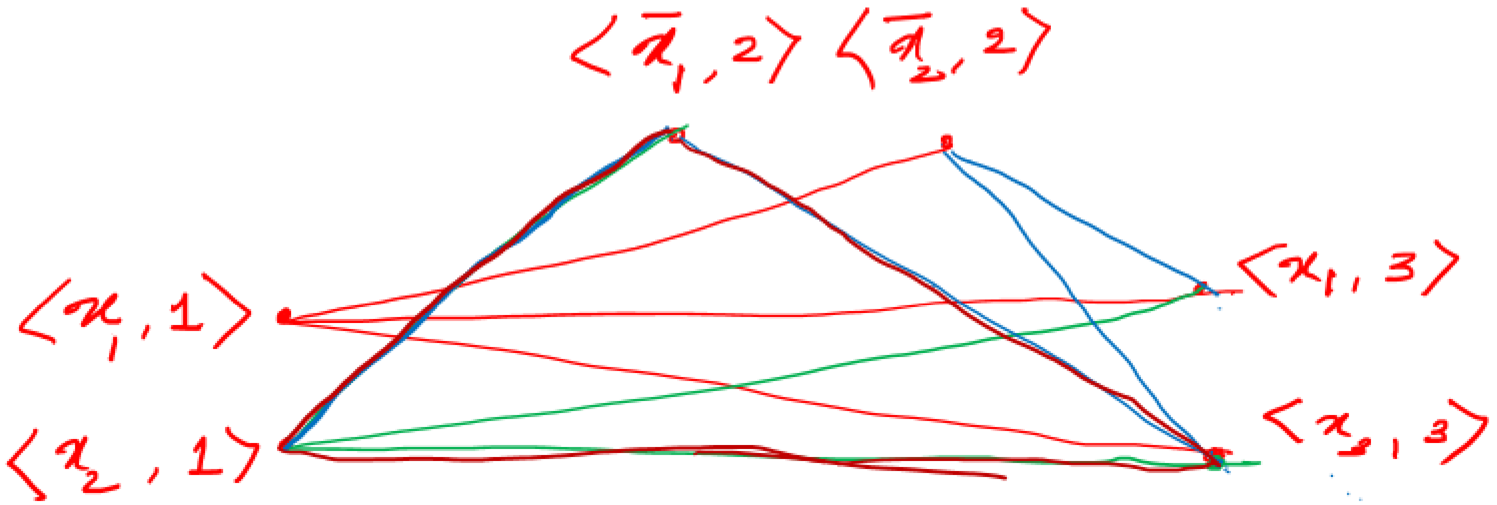
We can easily build this graph from ϕ in polynomial time. As an example of this construction, if we have

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

then G is the graph

$F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3) = 1$

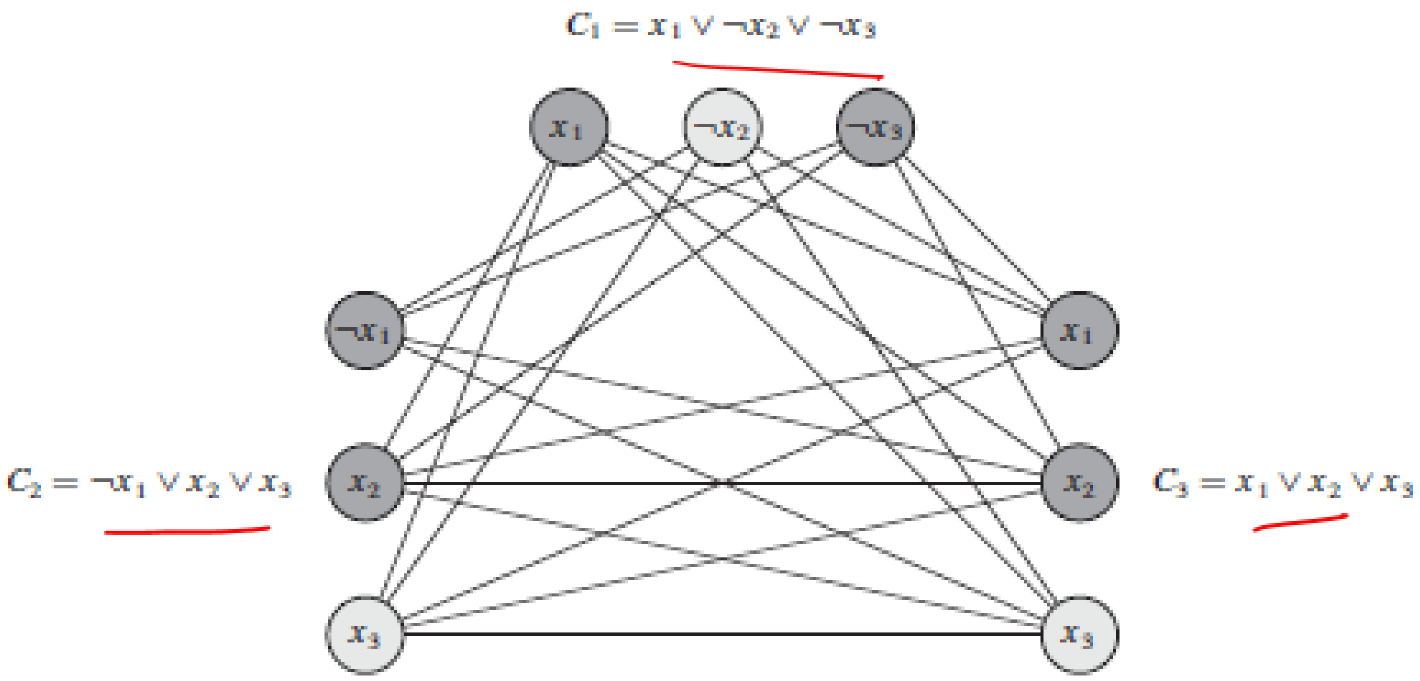
$\underline{\underline{k = 3}}$



$$\underbrace{\begin{array}{ccc} x_2, & \bar{x}_1, & x_3 \\ 1 & 1 & 1 \end{array}}$$

$$\left\{\begin{array}{l} x_2 = 1 \\ x_1 = 0 \\ x_3 = 1 \end{array}\right\} \checkmark\checkmark$$

The CLIQUE Problem


$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
$C_2 = \neg x_1 \vee x_2 \vee x_3$
$C_3 = x_1 \vee x_2 \vee x_3$
x_1
$\neg x_2$
$\neg x_3$
$\neg x_1$
x_2
x_3
x_1
x_2
x_3

The graph G derived from the 3-CNF formula $\phi = C_1 \wedge C_2 \wedge C_3$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$, and $C_3 = (x_1 \vee x_2 \vee x_3)$, in reducing 3-CNF-SAT to CLIQUE. A satisfying assignment of the formula has $x_2 = 0$, $x_3 = 1$, and x_1 either 0 or 1. This assignment satisfies C_1 with $\neg x_2$, and it satisfies C_2 and C_3 with x_3, corresponding to the clique with lightly shaded vertices.

The CLIQUE Problem Summary

- Given a 3-CNF formula F, we construct a graph G as follows.
- The graph has one node for each instance of each literal in the formula
- Two nodes are connected by an edge is:

(1) they correspond to literals in different clauses and

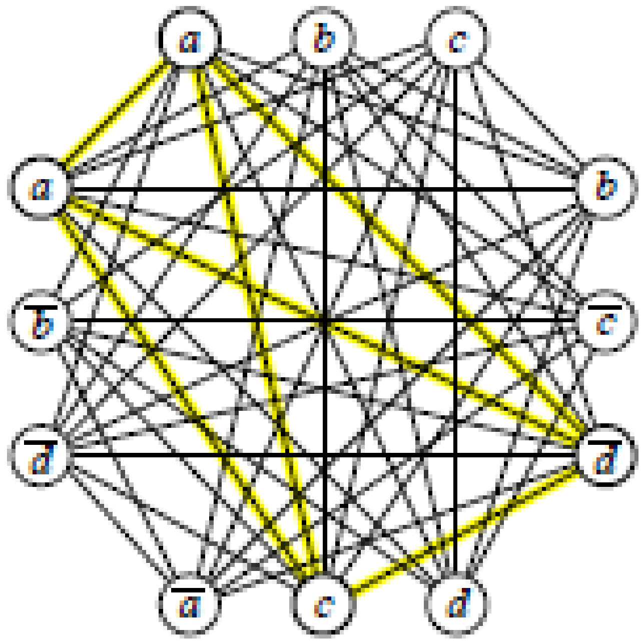
(2) those literals do not contradict each other

The CLIQUE Problem Summary

- Let F be the formula:

  $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

- This formula is transformed into the following graph:

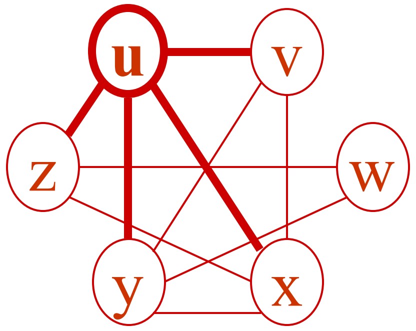


The CLIQUE Problem Summary

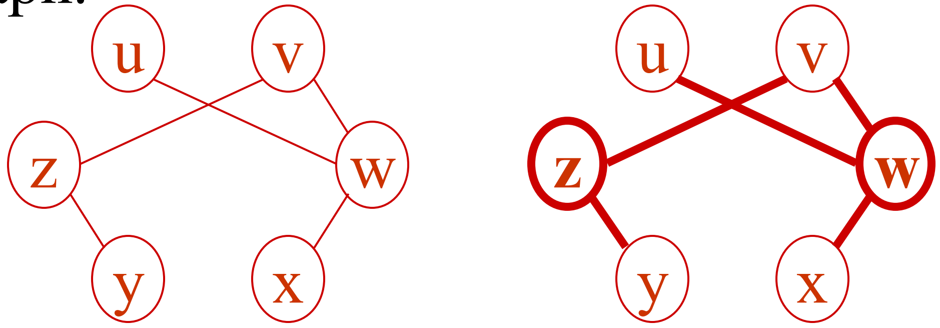
- Let F have k clauses. Then G has a clique of size k iff F has a satisfying assignment. The proof:
- **k-clique $\implies$ satisfying assignment:** If the graph has a clique of k vertices, then each vertex must come from a different clause. To get the satisfying assignment, we declare that each literal in the clique is true. Since we only connect non-contradictory literals with edges, this declaration assigns a consistent value to several of the variables. There may be variables that have no literal in the clique; we can set these to any value we like.
- **satisfying assignment $\implies$ k-clique:** If we have a satisfying assignment, then we can choose one literal in each clause that is true. Those literals form a k-clique in the graph.

The VERTEX-COVER Problem

A vertex **Covers** a set of edges:



A **Vertex-Cover** in an undirected graph is a set of vertices that cover all edges in the graph.



Vertex-Cover = {w ,z}

Vertex-Cover Problem

Optimization problem: Find a vertex-cover of minimum size in a graph.

Decision problem: Whether a graph has a vertex-cover of a given size k.

The VERTEX-COVER Problem

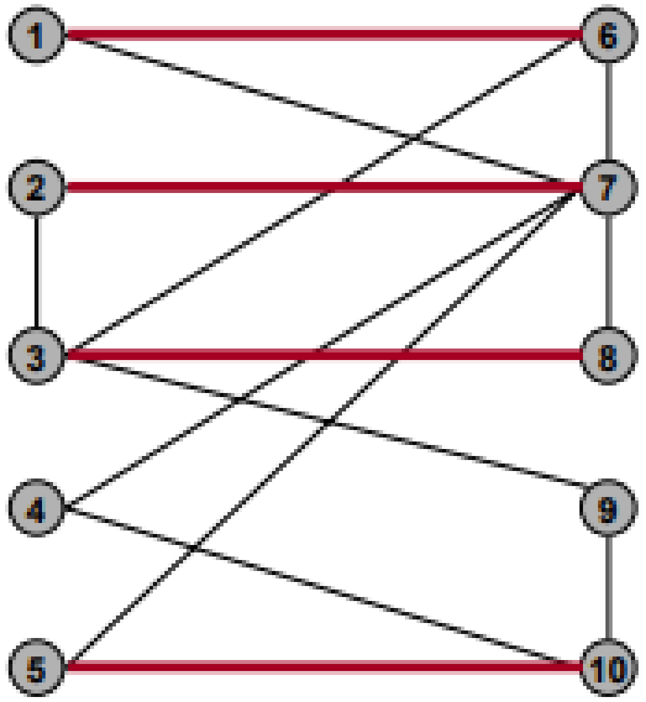
Ex.

- Is there a vertex cover of size 4?

  YES.

- Is there a vertex cover of size 3?

  NO.



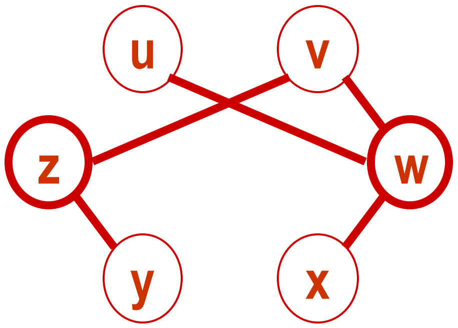
The VERTEX-COVER Problem

VERTEX-COVER is NP-complete

2 parts of the proof:

A. VERTEX-COVER $\in$NP

B. VERTEX-COVER is NP-hard



The VERTEX-COVER Problem

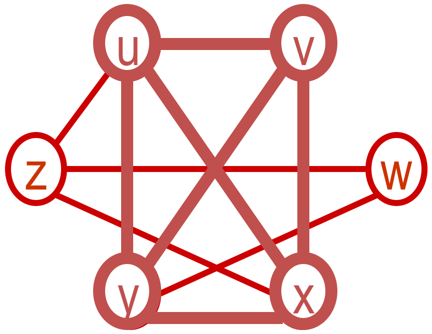
- <u>**VERTEX-COVER $\in$NP**</u>
- Consider an algorithm:

```
        bool Verify_VERTEX_COVER(Input_Graph,
                 Certificate /*a set of vertices*/  )
```

- This algorithm checks whether the set of vertices in the certificate cover all edges in the input graph.
- This is a 2-input, polynomial-time verification algorithm for **VERTEX-COVER**.
- Since we can find such an algorithm for **VERTEX-COVER**, we say that **VERTEX-COVER** can be verified in polynomial time, and **VERTEX-COVER** $\in$NP

The VERTEX-COVER Problem

- **<u>VERTEX-COVER is NP-hard</u>**
- Show that CLIQUE ≤p VERTEX-COVER
- ie. Any instance of CLIQUE satisfiability can be reduced in polynomial time to an instance of VERTEX-COVER.



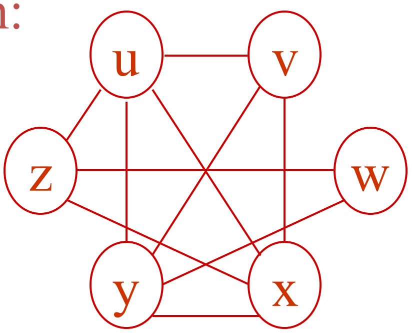
The VERTEX-COVER Problem

The Reduction :
For any CLIQUE satisfiability problem G=(V,E), k, we can create a VERTEX-COVER problem G',|V|-k such that G' is the ***complement*** of G.

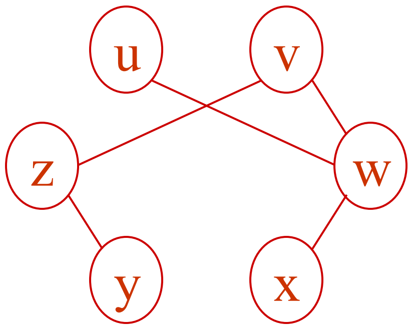
Example:

CLIQUE problem:
In the graph G=(V,E), can we find a clique of size k=4?

G



G' = ***complement*** of G



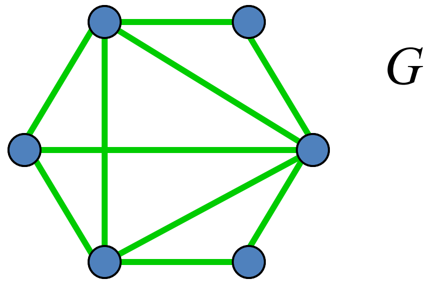
VERTEX-COVER problem:
In the graph G', can we find a clique of size |V|-k=2?

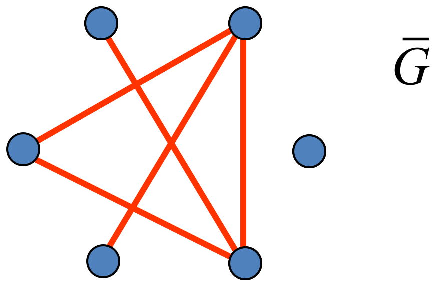
Complement of a graph

- Given an undirected graph G (V,E), we define the ***complement*** of G as G (V,E')where

  E'={(u,v): u,v $\in \boldsymbol{V, u \neq v \text{ and } (u,v) \notin}$ **E}**

- G is the graph containing exactly those edges that are not in G.
- The complement of *G* has all the edges that are missing in *G*— i.e. that would have to be added to make the complete graph.





The VERTEX-COVER Problem

- The reduction algorithm takes as input an instance (G,k) of the clique problem.
- It computes the complement G, which we can easily do in polynomial time.
- The output of the reduction algorithm is the instance (G’,|V|-k)of the vertex-cover problem.

The VERTEX-COVER Problem

Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$. We claim that $V - V'$ is a vertex cover in $\overline{G}$. Let (u, v) be any edge in $\overline{E}$. Then, $(u, v) \notin E$, which implies that at least one of u or v does not belong to V', since every pair of vertices in V' is connected by an edge of E. Equivalently, at least one of u or v is in $V - V'$, which means that edge (u, v) is covered by $V - V'$. Since (u, v) was chosen arbitrarily from $\overline{E}$, every edge of $\overline{E}$ is covered by a vertex in $V - V'$. Hence, the set $V - V'$, which has size $|V| - k$, forms a vertex cover for $\overline{G}$.