NP-Completeness, part II

Christian Wulff-Nilsen
Advanced Algorithms and Data Structures
DIKU

December 14, 2022

Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - SAT
  - 3-CNF-SAT
  - CLIQUE
  - VERTEX-COVER
  - (HAM-CYCLE)
  - TSP
  - SUBSET-SUM

Languages

- *Alphabet*: finite set Σ of symbols.
- *Language* L over alphabet Σ: a set of strings of symbols from Σ.

- We let $\Sigma = \{0, 1\}$ so L is a set of binary strings.
- Example: $L = \{0, 10, 11001, 11101, \ldots\}$.
- Σ^*: set of all binary strings (including ϵ).

Decision problems and languages

- A *decision problem* Q consists of yes-instances and no-instances.
- Example, $Q = \texttt{HAM-CYCLE}$: $\langle G \rangle$ is a yes-instance if G contains a simple cycle containing all vertices of G; otherwise $\langle G \rangle$ is a no-instance.
- We can view a problem Q as a mapping of yes-instances to 1 and no-instances to 0.
- We can also view Q as a language L:

$$L = \{x \in \{0,1\}^* | Q(x) = 1\}.$$

Verifying a language

- A *verification algorithm* is an algorithm A taking two arguments, $x, y \in \{0,1\}^*$, where y is the *certificate*.
- A *verifies* a string x if there is a certificate y such that $A(x,y) = 1$.
- The language verified by A is

$$L = \{x \in \{0,1\}^* | \text{there is a } y \in \{0,1\}^* \text{ such that } A(x,y) = 1\}.$$

The complexity class NP

- NP is the class of languages that can be verified in polynomial time.
- In other words, $L \in$ NP if and only if there is a polynomial-time verification algorithm A and a constant c such that

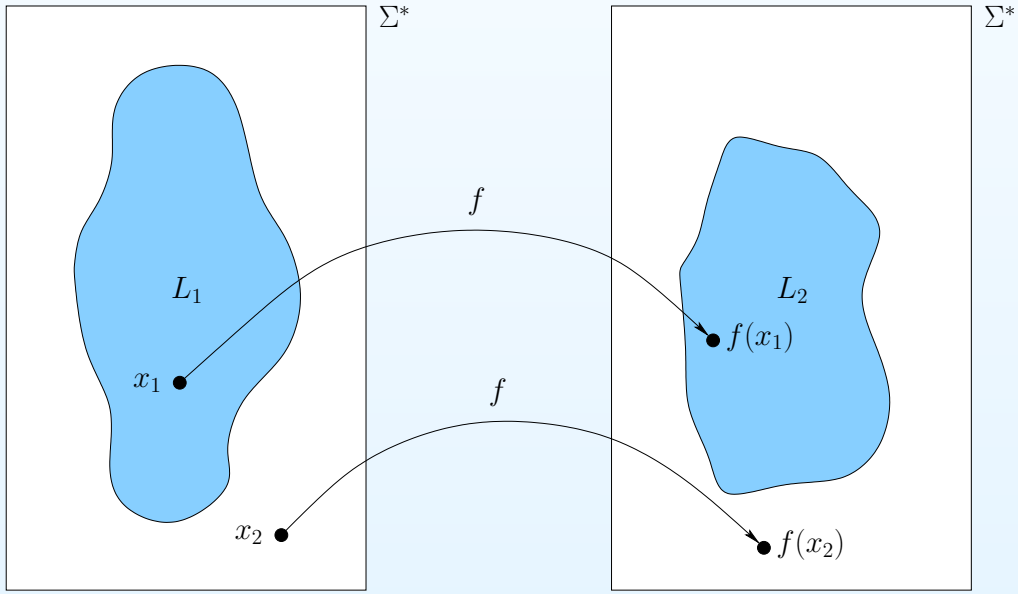
$$L = \{x \in \{0,1\}^* | \text{there is a } y \in \{0,1\}^* \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

- We saw that P $\subseteq$ NP.
- Big open problem: is P $=$ NP?

Reducibility

- Language L_1 is polynomial-time *reducible* to language L_2 if there is a polynomial-time computible function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$



- We use the notation $L_1 \leq_P L_2$ for this.
- We saw that

$$L_1 \leq_P L_2 \wedge L_2 \in P \Rightarrow L_1 \in P.$$

NP-completeness

- Language L is *NP-complete* if
  1. $L \in$ NP and
  2. $L' \leq_P L$ for every $L' \in$ NP.
- L is *NP-hard* if L satisfies property 2 (and possibly not property 1).
- We saw that if any language of NPC belongs to P then P $=$ NP.
- We also showed that `CIRCUIT-SAT` is NP-complete.

NP-completeness of other problems via reduction

- Let L and L' be two languages with $L' \in$ NPC.
- If $L' \leq_P L$ then L is NP-hard.
- If in addition $L \in$ NP then L is NP-complete.
- General technique to show NP-completeness of a language L:
  - Show that $L \in$ NP.
  - Pick another language L' known to be NP-complete (for instance, `CIRCUIT-SAT`).
  - Show that $L' \leq_P L$, i.e., show that there is a polynomial-time computible function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L' \Leftrightarrow f(x) \in L.$$

The SAT problem

- A *boolean formula* ϕ consists of boolean variables $x_1, \ldots, x_n$, boolean connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, and parentheses $($ and $)$.
- Example: $\phi = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$.
- A *satisfying assignment* for a boolean formula ϕ is an assignment of $0/1$-values to variables that makes ϕ evaluate to 1.
- ϕ is *satisfiable* if there exists a satisfying assignment for ϕ.
- We can now define the problem SAT:

$$\texttt{SAT} = \{\langle \phi \rangle | \phi \text{ is a satisfiable boolean formula}\}.$$

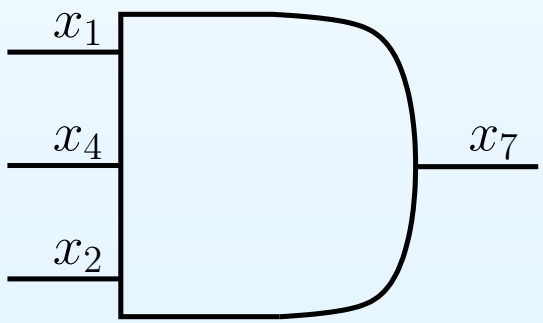
- We will show that SAT is NP-complete.

Showing that SAT is NP-complete

- To show $\texttt{SAT} \in$ NPC, we follow our recipe:
  - Show that $\texttt{SAT} \in$ NP.
  - Show that $\texttt{CIRCUIT-SAT} \leq_P \texttt{SAT}$.
- To show that $\texttt{SAT} \in$ NP, we construct a verification algorithm A taking inputs x and y.
- It regards x as a boolean formula ϕ and y as an assignment of values to variables of ϕ.
- A returns 1 if y defines a satisfying assignment for ϕ; otherwise, A returns 0.
- We can easily make A run in polynomial time.
- Thus, $\texttt{SAT} \in$ NP.

Showing CIRCUIT-SAT $\leq_P$ SAT

- Given a circuit C, we transform it into a boolean function ϕ as follows.
- Associate a variable x_i with each wire of C; let x_m be the output wire variable.
- We can view each gate of C as a function mapping the values on its input wires to the value on its output wire.
- Construct a sub-formula for each such function.
- Example:



Sub-formula for gate: $x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)$

Showing CIRCUIT-SAT $\leq_P$ SAT

- Example (see blackboard/Figure 34.10 in CLRS):

$$\begin{aligned}
\phi_1 &= (x_4 \leftrightarrow \neg x_3) \\
\phi_2 &= (x_5 \leftrightarrow (x_1 \vee x_2)) \\
\phi_3 &= (x_6 \leftrightarrow \neg x_4) \\
\phi_4 &= (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
\phi_5 &= (x_8 \leftrightarrow (x_5 \vee x_6)) \\
\phi_6 &= (x_9 \leftrightarrow (x_6 \vee x_7)) \\
\phi_7 &= (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)).
\end{aligned}$$

Showing $\texttt{CIRCUIT-SAT} \leq_P \texttt{SAT}$

- If $\phi_1, \ldots, \phi_k$ are the sub-formulas, we define ϕ to be $x_m \wedge \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_k$.
- Example:

$$\begin{aligned}\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ &\wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ &\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ &\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)).\end{aligned}$$

- ϕ can be constructed in polynomial time.
- In words, ϕ is stating that the output wire is 1 and that each gate behaves as it is supposed to.
- Thus, C is satisfiable if and only if ϕ is satisfiable:

$$\langle C \rangle \in \texttt{CIRCUIT-SAT} \Leftrightarrow \langle \phi \rangle \in \texttt{SAT}.$$

- We have now shown that $\texttt{SAT}$ is NP-complete.

3-CNF formulas

- Let ϕ be a boolean formula.
- A *literal* in ϕ is an occurence of a variable or its negation.
- Suppose ϕ is the AND of sub-formulas, called *clauses*.
- Furthermore, suppose that each clause is the OR of exactly 3 distinct literals.
- Then we say that ϕ is in 3-*conjunctive normal form*, or 3-*CNF*.
- Example:

$$\phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4).$$

NP-completeness of 3-CNF-SAT

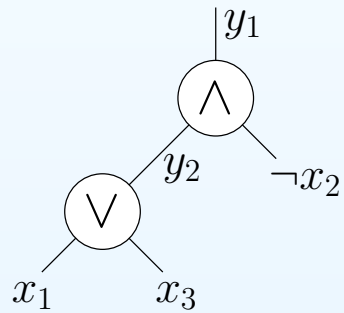
- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle\phi\rangle | \phi \text{ is in 3-CNF and satisfiable}\}.$$

- We will show that $3\text{-CNF-SAT} \in$ NPC in two steps:
  - $3\text{-CNF-SAT} \in$ NP,
  - $\text{SAT} \leq_P 3\text{-CNF-SAT}$.
- Showing $3\text{-CNF-SAT} \in$ NP is done using the same argument as for SAT.

NP-completeness of 3-CNF-SAT

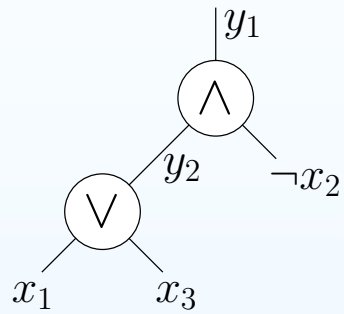
- Remains to show $\text{SAT} \leq_P 3\text{-CNF-SAT}$.
- Let $\langle \phi \rangle$ be an instance of SAT.
- We construct a *parse tree* for ϕ where leaves are literals and internal nodes are connectives.
- Parse tree for $\phi = (x_1 \vee x_3) \wedge \neg x_2$:



- We may assume that each node in the parse tree has at most two children. (why?)

NP-completeness of 3-CNF-SAT

- Parse tree for $\phi = (x_1 \vee x_3) \wedge \neg x_2$:



- Regard the tree as a circuit with internal nodes as gates.
- We can construct formulas $\phi'_1, \ldots, \phi'_k$ for each of these gates, as before.
- Let $\phi' = y_1 \wedge \phi'_1 \wedge \phi'_2 \wedge \ldots \wedge \phi'_k$, where y_1 is the output wire.
- ϕ' is satisfiable iff ϕ is satisfiable.
- Each clause of ϕ' has at most 3 literals.

NP-completeness of 3-CNF-SAT

- Consider one clause ϕ'_i.
- Example: $\phi'_i = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$.

y_1	y_2	x_2	ϕ'_i	$\phi''_i \equiv \neg\phi'_i$
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$\neg\phi'_i \equiv \phi''_i = (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).$$

NP-completeness of 3-CNF-SAT

- ϕ_i'' is in *disjunctive normal form*:

$$\phi_i'' = (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).$$

- Negating and applying De Morgan's laws converts ϕ_i'' into conjunctive normal form:

$$\neg\phi_i'' \equiv (y_1 \vee \neg y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2).$$

- Recall that $\phi_i'' \equiv \neg\phi_i'$.
- It follows that $\neg\phi_i'' \equiv \neg(\neg\phi_i') \equiv \phi_i'$.
- In words, $\neg\phi_i''$ is equivalent to the original ϕ_i'.

NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts ϕ' into an equivalent formula which is almost in 3-CNF:
  - each clause has *at most* 3 literals.
- We can extend this to *exactly* 3 (distinct) literals by introducing dummy variables p and q:

$$\ell_1 \vee \ell_2 \equiv (\ell_1 \vee \ell_2 \vee p) \wedge (\ell_1 \vee \ell_2 \vee \neg p)$$

$$\ell \equiv (\ell \vee p \vee q) \wedge (\ell \vee p \vee \neg q) \wedge (\ell \vee \neg p \vee q) \wedge (\ell \vee \neg p \vee \neg q)$$

- In polynomial time, we convert ϕ' into a formula ϕ''' in 3-CNF.
- We have

$$\langle \phi \rangle \in \texttt{SAT} \Leftrightarrow \langle \phi''' \rangle \in \texttt{3-CNF-SAT}.$$

- Thus, $\texttt{SAT} \leq_P \texttt{3-CNF-SAT}$ so $\texttt{3-CNF-SAT}$ is NP-complete.

The subset-sum problem

- Given a set S of positive integers and given integer target $t > 0$.
- Is there a subset S' of S summing to t?
- As a language:

$$\texttt{SUBSET-SUM} = \{\langle S, t\rangle | \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s\}.$$

- We will show that $\texttt{SUBSET-SUM}$ is NP-complete.
- Clearly, $\texttt{SUBSET-SUM} \in$ NP.
- To show NP-hardness, we reduce from $\texttt{3-CNF-SAT}$:

$$\texttt{3-CNF-SAT} \leq_P \texttt{SUBSET-SUM}.$$

Showing 3-CNF-SAT $\leq_P$ SUBSET-SUM

- Consider a 3-CNF-formula ϕ with n variables $x_1, \ldots, x_n$ and k clauses $C_1, \ldots, C_k$.
- We will construct an instance $\langle S, t \rangle$ such that:

$$\langle \phi \rangle \in 3\text{-CNF-SAT} \Leftrightarrow \langle S, t \rangle \in \text{SUBSET-SUM}.$$

- In other words, we want that ϕ is satisfiable if and only if S has a subset summing to t.

Constructing S and t (see blackboard/Figure 34.19 in CLRS)

- For each variable x_i, create two decimal numbers v_i and v'_i.
- For each clause C_j, create two decimal numbers s_j and s'_j.
- Finally, create a decimal number t.
- Each number has $n + k$ digits.
- The n most significant digits are associated with $x_1, \ldots, x_n$.
- The k least significant digits are associated with $C_1, \ldots, C_k$.
- S consists of numbers $v_1, v'_1, v_2, v'_2, \ldots, v_n, v'_n$ and $s_1, s'_1, s_2, s'_2, \ldots, s_k, s'_k$.

Constructing S and t (see blackboard/Figure 34.19 in CLRS)

- Digit x_i of v_i and digit x_i of v'_i is 1.
- Digit C_j of v_i is 1 if $x_i \in C_j$.
- Digit C_j of v'_i is 1 if $\neg x_i \in C_j$.
- Digit C_i of s_i is 1 and digit C_i of s'_i is 2.
- Target t is defined to be:

$$t = \overbrace{11\ldots1}^{n}\overbrace{44\ldots4}^{k}.$$

- All digits not specified above are 0.
- Simplifying assumptions: no clause contains both a variable and its negation and each variable appears somewhere. This ensures unique numbers (otherwise, S might be a multiset).

Showing $\langle \phi \rangle \in \texttt{3-CNF-SAT} \Rightarrow \langle S, t \rangle \in \texttt{SUBSET-SUM}$

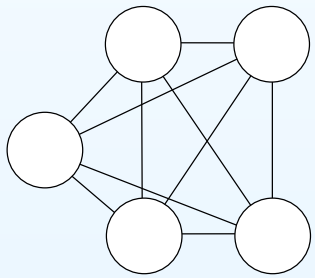
- Assume $\langle \phi \rangle \in \texttt{3-CNF-SAT}$.
- In other words, assume that ϕ is in 3-CNF and satisfiable.
- We will find a subset S' of S with $\sum_{s \in S'} s = t$.
- Assign values to $x_1, \ldots, x_n$ that satisfy ϕ.
- For $i = 1, \ldots, n$, if $x_i = 1$ then include v_i in S'; otherwise include v'_i.
- Include additional numbers from $\{s_1, s'_1, s_2, s'_2, \ldots, s_k, s'_k\}$ to reach target t.
- Why is this possible?

Showing $\langle S, t \rangle \in$ SUBSET-SUM $\Rightarrow \langle \phi \rangle \in$ 3-CNF-SAT

- Let S' be a subset of S summing to t.
- We need to find a satisfying assignment for ϕ.
- We set x_i to 1 if and only if $v_i \in S'$.
- Why is this a satisfying assignment?

The CLIQUE-problem

- Let $G = (V, E)$ be an undirected graph.
- A *clique* in G is a subset $V' \subseteq V$ such that $(u, v) \in E$ for all distinct $u, v \in V'$.
- The size of the clique is $|V'|$.
- Example of a clique of size 5:



- The CLIQUE problem is the problem of determining if G contains a clique of a given size k:

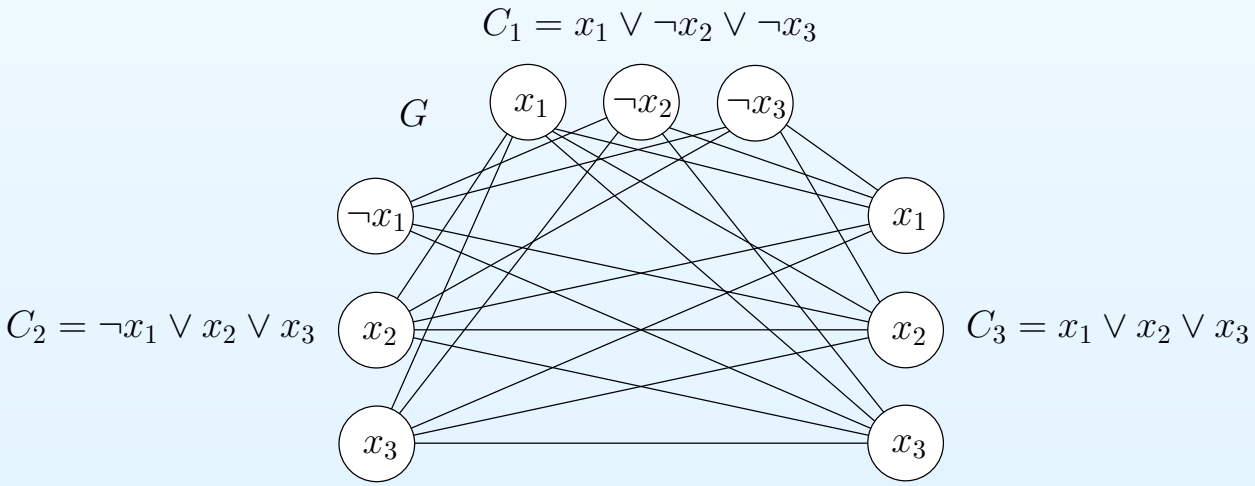
$$\texttt{CLIQUE} = \{\langle G, k\rangle | G \text{ is a graph containing a clique of size } k\}.$$

NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - CLIQUE $\in$ NP,
  - 3-CNF-SAT $\leq_P$ CLIQUE.
- To show CLIQUE $\in$ NP, consider an algorithm A taking two inputs, $\langle G, k \rangle$ and a certificate y.
- y specifies a subset V' of vertices of G.
- A checks that $|V'| = k$ and that V' is a clique in G.
- This can easily be done in polynomial time.
- Thus, CLIQUE $\in$ NP.

Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

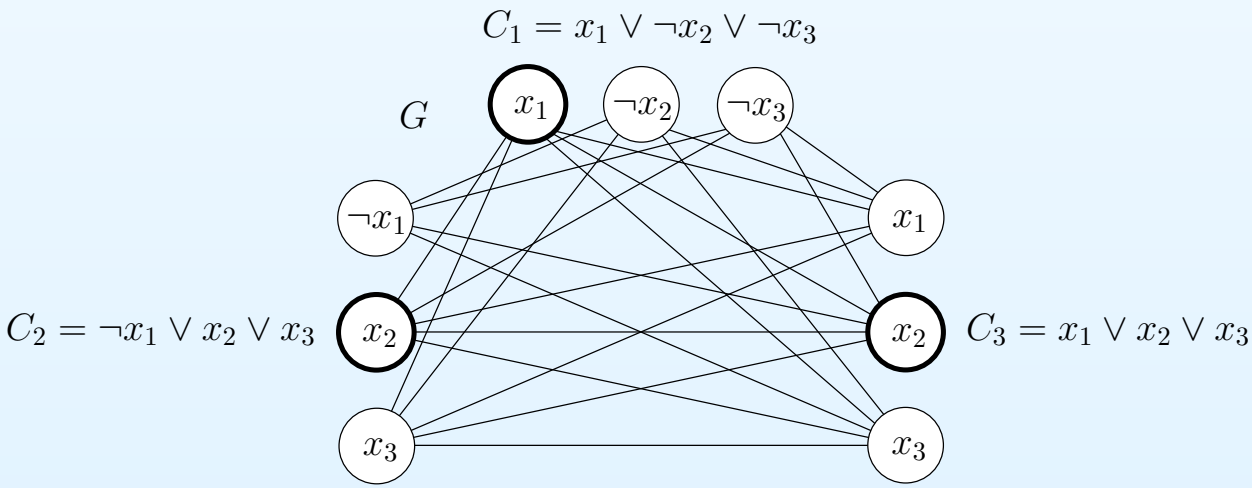
- Given a formula $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ in 3-CNF.
- We will construct a graph G such that ϕ is satisfiable if and only if G has a clique of size k.
- For each $C_r = \ell_1^r \vee \ell_2^r \vee \ell_3^r$, we include three vertices v_1^r, v_2^r, and v_3^r to G.
- There is an edge (v_i^r, v_j^s) in G if and only if $r \neq s$ and ℓ_i^r is not the negation of ℓ_j^s.
- Example with $\phi = C_1 \wedge C_2 \wedge C_3$:



- G can be constructed in polynomial time from ϕ.
- We will show $\langle \phi \rangle \in 3\text{-CNF-SAT} \Leftrightarrow \langle G, k \rangle \in \text{CLIQUE}$.

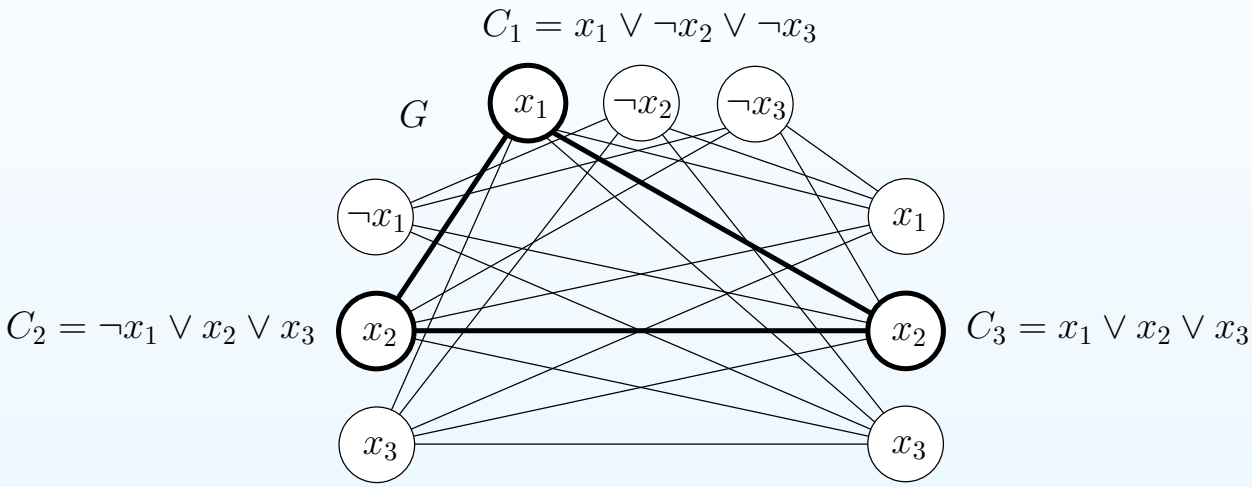
Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that ϕ is satisfiable if and only if G has a clique of size k.
- Assume first that $\langle \phi \rangle \in 3\text{-CNF-SAT}$.
- In a satisfying assignment for ϕ, each clause C_i of ϕ has at least one true literal.
- Pick the corresponding vertex in G.
- This gives a total of k vertices.
- Example with satisfying assignment $x_1 = 1, x_2 = 1, x_3 = 0$:



Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

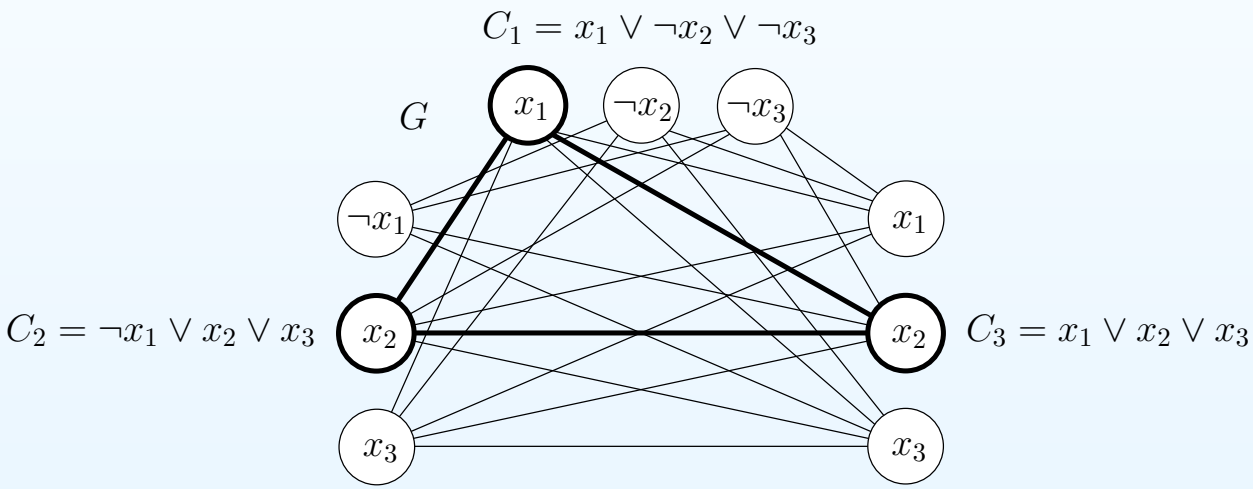
- Why is this a clique in G of size k?



- There must be an edge between each pair of these vertices since no picked literal can be the negation of another picked literal (we only picked true literals).
- Hence, G has a clique of size k so $\langle G, k \rangle \in \text{CLIQUE}$.

Showing 3-CNF-SAT $\leq_P$ CLIQUE

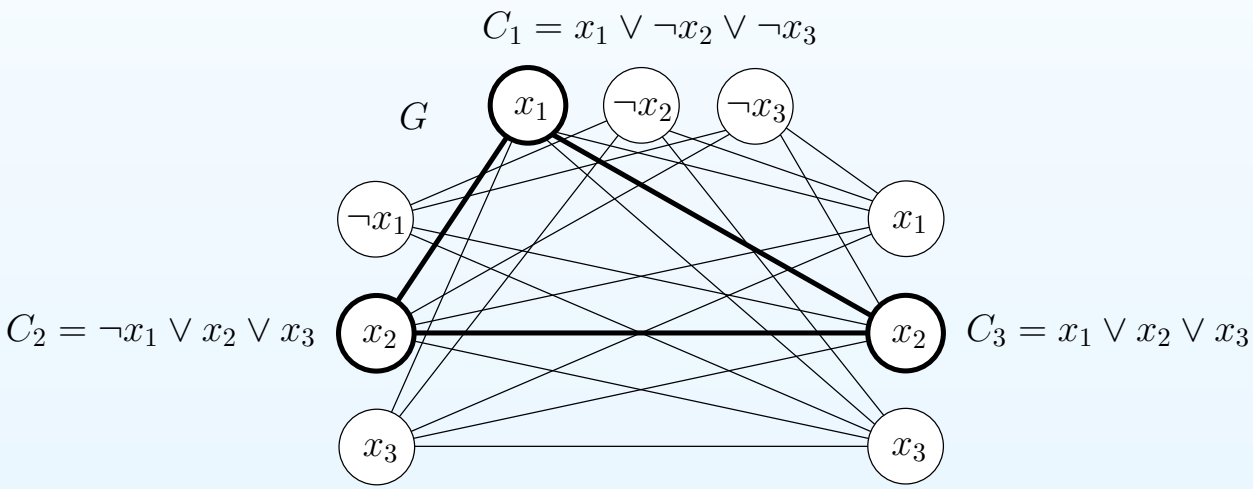
- It remains to show that if G has a clique of size k then ϕ is satisfiable.
- Let V' be a clique of G of size k.



- Each vertex triple of G has exactly one vertex in V'.
- We assign 1 to the literal of ϕ corresponding to that vertex.
- In the example above, $x_1 = 1$ and $x_2 = 1$.

Showing 3-CNF-SAT $\leq_P$ CLIQUE

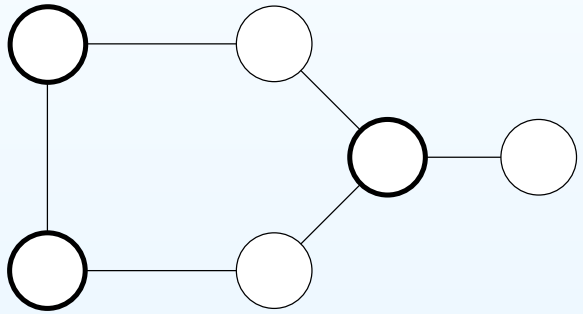
- It remains to show that if G has a clique of size k then ϕ is satisfiable.
- Let V' be a clique of G of size k.



- No variable of ϕ is assigned both 0 and 1 in this way since there is no edge between a variable and its negation in G.
- This gives a legal assignment of values to some of the variables.
- This assignment satisfies ϕ as it makes each clause true.
- What about variables that have not been assigned a value? (x_3 in the example) Pick an arbitrary value for each of them.

The VERTEX-COVER problem

- A *vertex cover* of $G = (V, E)$ is a subset $V' \subseteq V$ such that every edge of E has at least one endpoint in V'.
- Vertex cover problem: find a minimum-size vertex cover of G.



- Restating as a decision problem: does G have a vertex cover of a given size k?
- Stated as a language:

$$\texttt{VERTEX-COVER} = \{\langle G, k\rangle | G \text{ has a vertex cover of size } k\}.$$

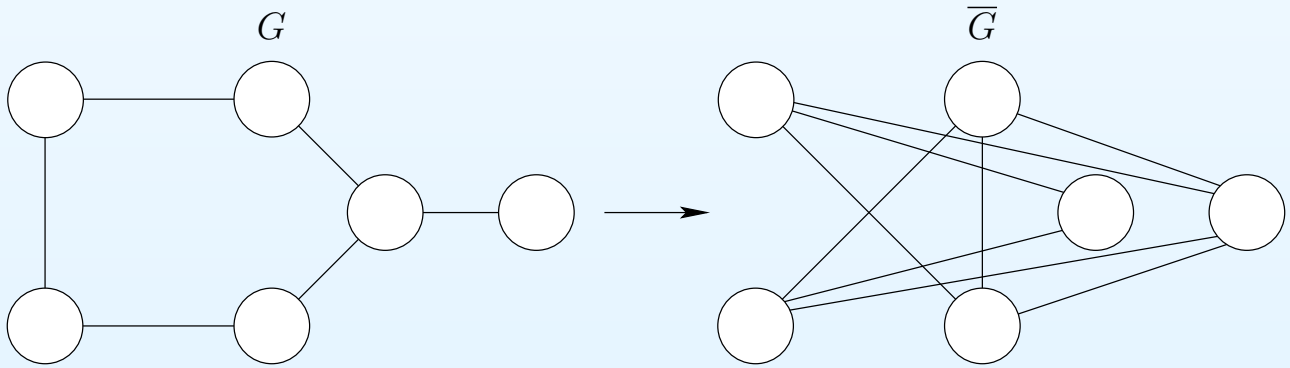
- We will show that VERTEX-COVER is NP-complete.

Showing that VERTEX-COVER $\in$ NP

- The verification algorithm takes an instance $\langle G, k \rangle$ and a certificate denoting a subset V' of vertices of G.
- It checks that V' has size k and that every edge of G is incident to at least one vertex of V'.
- This can easily be done in polynomial time.
- Hence, VERTEX-COVER $\in$ NP.

Showing that VERTEX-COVER is NP-hard

- We show CLIQUE $\leq_P$ VERTEX-COVER.
- Given an instance $\langle G, k \rangle$ of the clique problem.
- Let n denote the number of vertices of G.
- We transform $\langle G, k \rangle$ in polynomial time to the instance $\langle \overline{G}, n - k \rangle$ of the vertex cover problem.
- Here, $\overline{G}$ is the *complement* of G which has the same vertex set as G and has an edge between two vertices u and v if and only if there is no edge between u and v in G.

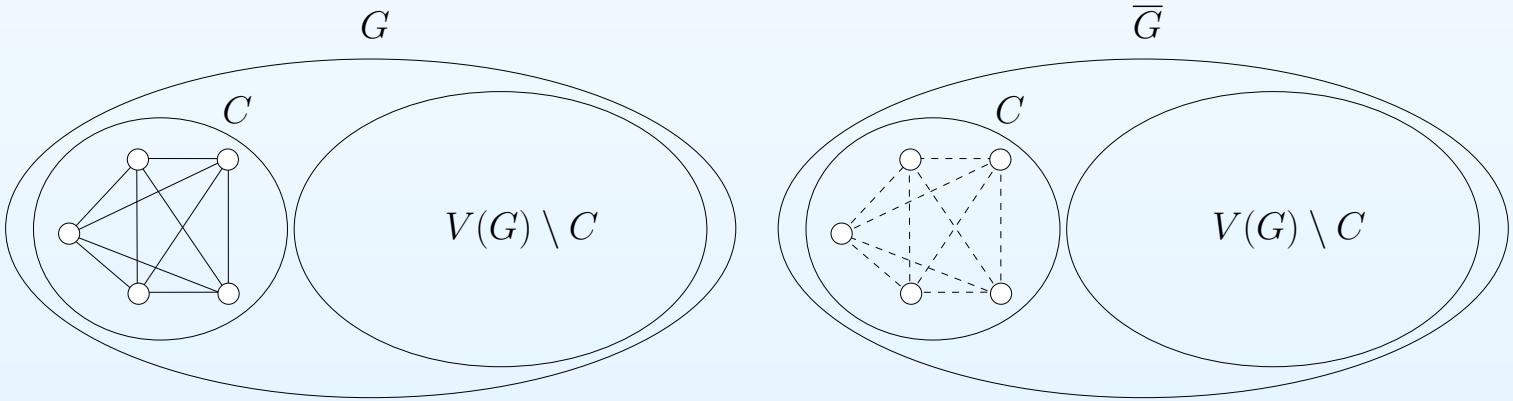


Showing that VERTEX-COVER is NP-hard

- Need to show:

$$\langle G, k \rangle \in \texttt{CLIQUE} \Leftrightarrow \langle \overline{G}, n - k \rangle \in \texttt{VERTEX-COVER}.$$

- Observe that any vertex set C is a clique in G if and only if C contains no edges in $\overline{G}$.



- Hence, C is a clique of size k in G if and only if $V(G) \setminus C$ is a vertex cover of size $n - k$ in $\overline{G}$.
- We have shown that VERTEX-COVER is NP-hard.

The travelling-salesman problem

- Let $G = (V, E)$ be a complete undirected graph and let c be a non-negative integer cost function on the edge set of G.
- A *tour* of G is a Hamilton cycle of G.
- The travelling salesman problem is that of finding a minimum-cost tour of G.
- Stated as a decision problem/language:

$$\texttt{TSP} = \{\langle G, c, k\rangle | G \text{ has a tour of cost at most } k\}.$$

- Clearly, $\texttt{TSP} \in$ NP.
- Since HAM-CYCLE is NP-complete, we show that TSP is NP-hard by:

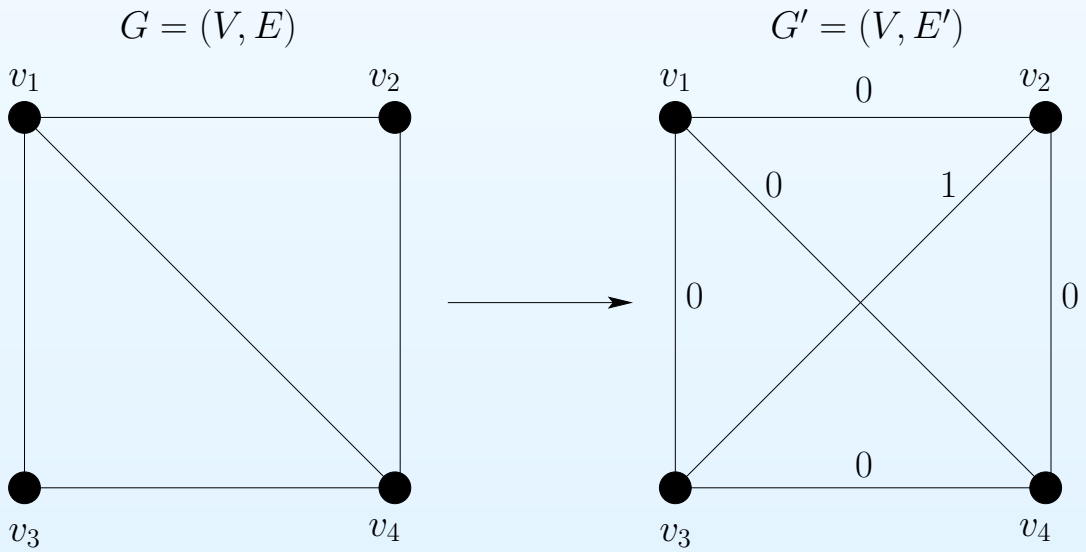
$$\texttt{HAM-CYCLE} \leq_P \texttt{TSP}.$$

- This will imply that TSP is NP-complete.

HAM-CYCLE $\leq_P$ TSP

- Let $G = (V, E)$ be an instance of the Hamilton-cycle problem.
- We construct a complete graph $G' = (V, E')$ on vertex set V.
- Define a cost function c on E' by

$$c(u, v) = \begin{cases} 0 & \text{if } (u, v) \in E, \\ 1 & \text{if } (u, v) \notin E. \end{cases}$$



- Show that: $\langle G \rangle \in$ HAM-CYCLE $\Leftrightarrow \langle G', c, 0 \rangle \in$ TSP.