# 2SAT, Independent Set and Vertex Cover

## CSCI3360 Introduction to Computational Complexity
## Tutorial (Week 7)

Chak Fai Yung

Department of Computer Science and Engineering
The Chinese University of Hong Kong

# Today's tutorial

1. **Section 4.6.2: 2SAT.** Implication graphs, strongly connected components, and a satisfying assignment.
2. **A brief complexity recap.** NP-hardness, NP-completeness, and the direction of a reduction.
3. **Section 5.1: Independent set.** A reduction from CLIQUE.
4. **Section 5.2: Vertex cover.** The decision problem and its NP-completeness proof.

# 2SAT: a restricted satisfiability problem

A **literal** is a variable $x$ or its negation $\neg x$.

## 2-CNF and the decision problem

A 2-CNF formula is a conjunction of clauses, each containing two literals:

$$\varphi = \bigwedge_{j=1}^{m} (a_j \vee b_j).$$

**2SAT asks:** does some assignment make every clause true?

For example,

$$\varphi = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$$

is satisfied by $x = y = \text{true}$.

We will derive a graph algorithm and prove that $2\text{SAT} \in \text{P}$.

# A clause becomes implications

For any literals $a, b$,

$$\boxed{(a \vee b) \equiv (\neg a \Rightarrow b) \wedge (\neg b \Rightarrow a).}$$

| $a$ | $b$ | $a \vee b$ | $\neg a \Rightarrow b$ | $\neg b \Rightarrow a$ |
|---|---|---|---|---|
| F | F | F | F | F |
| F | T | T | T | T |
| T | F | T | T | T |
| T | T | T | T | T |

An implication fails only when its premise is true and its conclusion is false.

**Interpretation:** if one literal in a clause is false, the other must be true.

# Build the implication graph

For $n$ variables and $m$ clauses, construct a directed graph $G_\varphi$:

1. Create **two vertices per variable**: $x_i$ and $\neg x_i$.
2. For every clause $(a \vee b)$, add **both** directed edges

$$\neg a \longrightarrow b, \qquad \neg b \longrightarrow a.$$




The graph has $2n$ vertices and at most $2m$ distinct edges.

# Running example: list all six implications

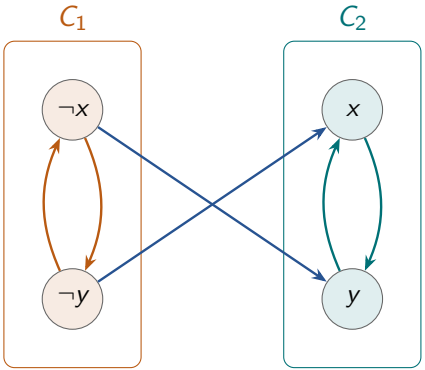
$$\varphi = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

| Clause | First implication | Second implication |
|---|---|---|
| $(x \vee y)$ | $\neg x \to y$ | $\neg y \to x$ |
| $(\neg x \vee y)$ | $x \to y$ | $\neg y \to \neg x$ |
| $(x \vee \neg y)$ | $\neg x \to \neg y$ | $y \to x$ |

The positive literals reach one another, and the negative literals reach one another.

There are also edges from negative literals to positive literals.

# Running example: two strongly connected groups




$$C_1 = \{\neg x, \neg y\},$$
$$C_2 = \{x, y\}.$$

Inside each group, every vertex can reach every other vertex.

Edges go from $C_1$ to $C_2$; no path returns to $C_1$.

# Directed paths propagate truth

If $a \to b$ is an edge, every satisfying assignment obeys

$$a = \text{true} \implies b = \text{true}.$$

Therefore a directed path

$$a = \ell_0 \to \ell_1 \to \cdots \to \ell_t = b$$

means that making $a$ true forces $b$ true.

**Why paths matter**

An individual edge records one clause. A path combines the consequences of several clauses.

For the running example, $\neg x \to \neg y \to x$. Thus $\neg x$ cannot be true in a satisfying assignment.

# SCCs and the condensation DAG

A **strongly connected component (SCC)** is a maximal set $C$ such that, for all $u, v \in C$,

$$u \rightsquigarrow v \quad \text{and} \quad v \rightsquigarrow u.$$

Contract every SCC to one vertex. Retain edges between different components. The result is the **condensation graph**, a directed acyclic graph (DAG).




**Why acyclic?** A directed cycle of components would make them mutually reachable, so they would be one SCC.

# The criterion that decides 2SAT

**SCC characterization**

A 2-CNF formula $\varphi$ is satisfiable if and only if

$$C(x_i) \neq C(\neg x_i) \qquad \text{for every variable } x_i,$$

where $C(\ell)$ denotes the SCC containing literal $\ell$.

We must prove **both directions**:

1. If $x$ and $\neg x$ share an SCC, no assignment works.
2. If every pair is separated, construct an assignment that satisfies every clause.

The second direction needs an explicit rule; separation alone is not yet a proof.

# Proof: one SCC containing $x$ and $\neg x$ is impossible

Suppose $C(x) = C(\neg x)$. There are paths in **both** directions:

$$x \rightsquigarrow \neg x, \qquad \neg x \rightsquigarrow x.$$

- If $x = \mathrm{true}$, the first path forces $\neg x = \mathrm{true}$: a contradiction.
- If $x = \mathrm{false}$, then $\neg x = \mathrm{true}$, and the second path forces $x = \mathrm{true}$: a contradiction.

**Conclusion**

Neither truth value for $x$ can occur in a satisfying assignment. Hence $\varphi$ is unsatisfiable.

# A symmetry we need for the converse

Every implication edge has a **contrapositive partner**:

$$a \to b \quad \Longrightarrow \quad \neg b \to \neg a.$$

Indeed, the two edges are generated by the same clause $(\neg a \vee b)$.

Reversing a path and negating every literal gives

$$a \rightsquigarrow b \quad \Longrightarrow \quad \neg b \rightsquigarrow \neg a.$$

Thus if $a$ and $b$ are in the same SCC, so are $\neg a$ and $\neg b$.

**Paired components**

Negating all literals of an SCC gives another SCC. Under our assumption, a component and its negated component are distinct.

# Construct an assignment from a topological order

Assume $C(x) \neq C(\neg x)$ for every variable. Topologically order the condensation DAG, with ranks **increasing along edges**:

$$C \to D \quad \Longrightarrow \quad r(C) < r(D).$$

Assignment rule: the later component is true

$$x = \text{true} \quad \Longleftrightarrow \quad r(C(x)) > r(C(\neg x)).$$

The two ranks are distinct, so this chooses exactly one truth value for each variable.

Equivalently, any literal $a$ is true precisely when $r(C(a)) > r(C(\neg a))$.

# Proof: the rule respects every implication

Write $r(a) = r(C(a))$ for brevity. For an edge $a \to b$, we also have $\neg b \to \neg a$, so

$$r(a) \le r(b), \qquad r(\neg b) \le r(\neg a).$$

Suppose this edge were violated: $a$ is true and $b$ is false. The assignment rule would give

$$r(\neg a) < r(a), \qquad r(b) < r(\neg b).$$

Combining all four inequalities,

$$\underbrace{r(\neg a) < r(a) \le r(b) < r(\neg b) \le r(\neg a)}_{\text{impossible}}.$$

Therefore **no edge goes from a true literal to a false literal**.

# Proof: every clause is satisfied

Consider an arbitrary clause $(a \vee b)$. Its implication graph contains $\neg a \to b$.

If the clause were false, then $a$ and $b$ would both be false. Hence

$$\neg a = \text{true}, \qquad b = \text{false}.$$

This would be a true-to-false implication edge, which the previous slide ruled out.

**Conclusion of the converse**

The constructed assignment satisfies every clause. Thus $\varphi$ is satisfiable whenever every variable is separated from its negation.

Together with the first direction, this proves the SCC characterization.

# The complete 2SAT algorithm

1. Build the implication graph on $x_1, \neg x_1, \ldots, x_n, \neg x_n$.
2. Compute all SCCs.
3. For each variable $x_i$, compare $C(x_i)$ and $C(\neg x_i)$.
   - If any pair agrees, return **no**.
4. Otherwise return **yes**. To output an assignment as well:
   - Build the condensation DAG and topologically order its vertices.
   - Set $x_i$ true iff $r(C(x_i)) > r(C(\neg x_i))$.

**Correctness:** Step 3 detects a contradiction; Step 4 uses the constructive proof.

# Why the algorithm takes linear time

Let $n$ be the number of variables and $m$ the number of clauses.

| Task | Time with adjacency lists |
|---|---|
| Create $2n$ vertices and $2m$ edge occurrences | $O(n+m)$ |
| Compute SCCs | $O(n+m)$ |
| Check every variable against its negation | $O(n)$ |
| Build and topologically order condensation DAG | $O(n+m)$ |
| Read off the assignment | $O(n)$ |

Total time $O(n+m)$, $\qquad 2\mathrm{SAT} \in \mathrm{P}$.

By contrast, $3\mathrm{SAT}$ is NP-complete. Allowing a third literal changes the available structure.

# Running example: recover and check the assignment

$$\varphi = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

The condensation graph is

$$\underbrace{\{\neg x, \neg y\}}_{r=1} \longrightarrow \underbrace{\{x, y\}}_{r=2}.$$

Both positive literals have higher ranks than their negations, so

$$x = \text{true}, \qquad y = \text{true}.$$

Substitution verifies all three clauses:

$$(\mathrm{T} \vee \mathrm{T}) \wedge (\mathrm{F} \vee \mathrm{T}) \wedge (\mathrm{T} \vee \mathrm{F}) = \mathrm{T}.$$

**Check the direction:** setting the earlier component true would violate the edges to the later component.

# An unsatisfiable example: exhibit the two paths

Consider

$$\psi = (x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y).$$

The graph contains the following paths, each justified by two clauses:

$$x \longrightarrow y \longrightarrow \neg x \qquad \text{from } (\neg x \vee y),\ (\neg x \vee \neg y),$$

$$\neg x \longrightarrow y \longrightarrow x \qquad \text{from } (x \vee y),\ (x \vee \neg y).$$

Thus $x$ and $\neg x$ lie in the same SCC, so the algorithm returns **no**.

**Direct check:** if $x$ is false, the first two clauses require both $y$ and $\neg y$; if $x$ is true, the last two do.

# Practice: apply the criterion

Let

$$\theta = (\neg p \vee q) \wedge (\neg q \vee p) \wedge (\neg p \vee \neg q).$$

1. Write both implication edges for each clause.
2. Find the SCCs and draw the condensation DAG.
3. Is the formula satisfiable? Use topological ranks to produce an assignment.
4. A graph has a path $p \rightsquigarrow \neg p$, but no path back. Does that alone prove unsatisfiability?

For Question 4, distinguish **one forced truth value** from a contradiction that rules out **both** truth values.

# Practice: solution and a common mistake

| Clause | First edge | Second edge |
|---|---|---|
| $(\neg p \vee q)$ | $p \to q$ | $\neg q \to \neg p$ |
| $(\neg q \vee p)$ | $q \to p$ | $\neg p \to \neg q$ |
| $(\neg p \vee \neg q)$ | $p \to \neg q$ | $q \to \neg p$ |

The SCCs and their ranks are

$$\underbrace{\{p, q\}}_{r=1} \longrightarrow \underbrace{\{\neg p, \neg q\}}_{r=2}.$$

So $p = q = \text{false}$; substitution makes every clause true.

**A one-way path is not enough to reject**

$p \rightsquigarrow \neg p$ forces $p$ false in any satisfying assignment. Only mutual reachability $p \rightsquigarrow \neg p \rightsquigarrow p$ rules out both truth values.

# NP-hard and NP-complete

**Recall NP:** an input is a yes-instance if and only if it has a polynomial-size certificate accepted by a deterministic polynomial-time verifier.

### NP-hardness

A problem $B$ is **NP-hard** if every language $L \in \mathrm{NP}$ reduces to $B$ in polynomial time:

$$\forall L \in \mathrm{NP}, \qquad L \leq_P B.$$

### NP-completeness

$B$ is **NP-complete** if $B \in \mathrm{NP}$ and $B$ is NP-hard.

If an NP-complete problem has a polynomial-time algorithm, then $\mathrm{P} = \mathrm{NP}$.

# The direction of a polynomial-time reduction

$$A \leq_P B \iff \exists \text{ polynomial-time } f: \quad x \in A \iff f(x) \in B.$$

An algorithm for $B$ can then solve $A$:

$$x \xrightarrow{\text{construct } f} f(x) \xrightarrow{\text{algorithm for } B} \text{answer for } A$$

To prove a new problem $B$ is NP-hard

Choose a known NP-hard problem $A$ and prove $\boxed{A \leq_P B}$.

The construction must preserve both yes-instances and no-instances.

# Why one reduction transfers NP-hardness

Suppose $A$ is NP-hard and $A \leq_P B$. For an arbitrary $L \in \mathrm{NP}$,

$$L \leq_P A \leq_P B.$$

If $f$ reduces $L$ to $A$ and $g$ reduces $A$ to $B$, then

$$x \in L \iff f(x) \in A \iff g(f(x)) \in B.$$

The composition $g \circ f$ is polynomial-time, so $B$ is NP-hard.

The proof checklist we will use

1. Give a certificate and verifier to prove $B \in \mathrm{NP}$.
2. Reduce a known NP-complete problem to $B$.
3. Prove both correctness directions and polynomial running time.

# Cook–Levin and the NP-completeness of 3SAT

**Cook–Levin theorem (Theorem 4.5.1)**

$\mathrm{SAT}$ is NP-complete.

Here SAT asks whether a CNF formula has a satisfying truth assignment.

**Recall the conversion:** repeat literals in short clauses and introduce fresh auxiliary variables for long clauses. This gives

$$\mathrm{SAT} \leq_P 3\mathrm{SAT}.$$

The construction takes polynomial time and preserves satisfiability.

Since a proposed assignment for a 3-CNF formula can be checked in polynomial time, $3\mathrm{SAT} \in \mathrm{NP}$. Thus **3SAT is NP-complete** (Corollary 4.6.1).

# Recall: CLIQUE is NP-complete

$$\mathrm{CLIQUE} = \{\langle G, k\rangle : G \text{ has a clique of size at least } k\}.$$

**Membership in NP:** list $k$ distinct vertices of $G$ and check that every pair is adjacent.

**NP-hardness:** last week's reduction established

$$3\mathrm{SAT} \leq_P \mathrm{CLIQUE}.$$

For a formula with $m$ clauses, use one vertex per literal occurrence, connect compatible occurrences from different clauses, and set $k = m$.

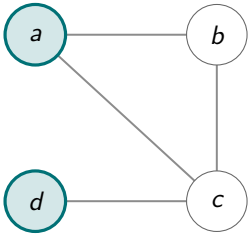
Corollary 4.6.2

Since 3SAT is NP-complete and CLIQUE belongs to NP, **CLIQUE is NP-complete**.

# Independent set

An **independent set** in $G = (V, E)$ is a set $S \subseteq V$ with no edge between two vertices of $S$.

$$\text{INDEPENDENT-SET} = \{\langle G, k\rangle : G \text{ has an independent set of size at least } k\}.$$



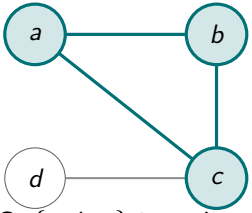

$S = \{a, d\}$ is independent.

$\{a, c\}$ is not independent, since $\{a, c\} \in E$.

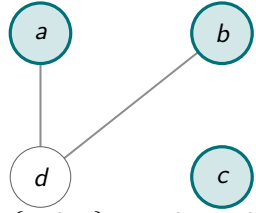
# The complement graph

The complement $\overline{G} = (V, \overline{E})$ keeps the vertices and reverses adjacency:

$$\{u, v\} \in \overline{E} \quad \Longleftrightarrow \quad \{u, v\} \notin E \qquad (u \neq v).$$




$G$: $\{a, b, c\}$ is a clique




$\overline{G}$: $\{a, b, c\}$ is independent

# Clique and independent set

Fix a set $S \subseteq V$. Then

$$\begin{aligned} S \text{ is a clique in } G &\iff \text{every distinct } u, v \in S \text{ are adjacent in } G \\ &\iff \text{no distinct } u, v \in S \text{ are adjacent in } \overline{G} \\ &\iff S \text{ is independent in } \overline{G}. \end{aligned}$$

The reduction

$$f(\langle G, k \rangle) = \langle \overline{G}, k \rangle.$$

The graph changes. The threshold $k$ stays the same.

# Independent-set certificates and verification

For an input $\langle G, k \rangle$ with $0 \le k \le n = |V|$:

1. A certificate lists exactly $k$ vertex identifiers.
2. Check that every listed vertex belongs to $V$ and no identifier repeats.
3. Check that each of the $\binom{k}{2}$ pairs is a nonedge of $G$.

With an adjacency matrix, the pair checks take $O(k^2)$ time. The certificate has polynomial length.

**Why the verifier characterizes the language**

A set with at least $k$ independent vertices contains a set of exactly $k$ independent vertices. An accepted list is itself such a set.

Therefore $\text{INDEPENDENT-SET} \in \text{NP}$.

# CLIQUE to independent set: both directions

Given a CLIQUE instance $\langle G, k \rangle$, output $\langle \overline{G}, k \rangle$.

**Forward.** Suppose $G$ has a clique $S$ with $|S| \geq k$. Every pair of vertices in $S$ is an edge of $G$, so no such pair is an edge of $\overline{G}$. Thus $S$ is independent in $\overline{G}$ with the same size.

**Reverse.** Suppose $\overline{G}$ has an independent set $S$ with $|S| \geq k$. No pair in $S$ is an edge of $\overline{G}$, so every distinct pair is an edge of $G$. Thus $S$ is a clique in $G$.

$$\langle G, k \rangle \in \text{CLIQUE} \quad \Longleftrightarrow \quad \langle \overline{G}, k \rangle \in \text{INDEPENDENT-SET}.$$

# Polynomial construction and NP-completeness

To build $\overline{G}$:

1. Keep the vertex set $V$.
2. For every unordered pair $\{u, v\}$ with $u \neq v$, include it exactly when $\{u, v\} \notin E(G)$.
3. Keep the threshold $k$.

There are $\binom{n}{2}$ pairs, so a matrix-based construction takes $O(n^2)$ time and produces polynomial-size output.

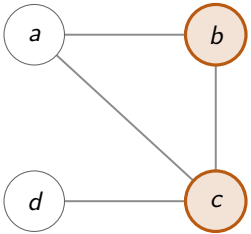
Theorem 5.1.1

INDEPENDENT-SET $\in$ NP and CLIQUE $\leq_P$ INDEPENDENT-SET. Since CLIQUE is NP-complete, INDEPENDENT-SET is NP-complete.

# Vertex cover

A **vertex cover** $C \subseteq V$ contains at least one endpoint of every edge.

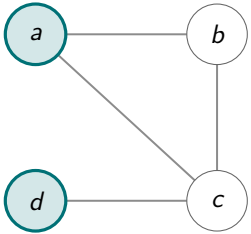
$$\text{VERTEX-COVER} = \{\langle G, k\rangle : G \text{ has a vertex cover of size at most } k\}.$$



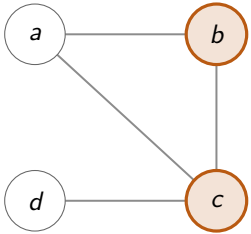

$C = \{b, c\}$ covers every edge.

An edge may have **both** endpoints in $C$.

# An independent set and its complementary cover




$S = \{a, d\}$




$V \setminus S = \{b, c\}$

**Lemma 5.2.1**

$S$ is independent in $G$ if and only if $V \setminus S$ is a vertex cover of $G$.

# Why the complementary set covers the edges

**Forward direction.** Suppose $S$ is independent.

- No edge has both endpoints in $S$.
- Every edge therefore has an endpoint in $V \setminus S$.

So $V \setminus S$ is a vertex cover.

**Reverse direction.** Suppose $V \setminus S$ is a vertex cover.

- If two vertices in $S$ were adjacent, neither endpoint of that edge would lie in the cover.
- This contradicts the cover property.

So $S$ is independent.

# Independent set to vertex cover: the threshold

Let $n = |V|$. Given $\langle G, k \rangle$ for INDEPENDENT-SET, output

$$\boxed{\langle G, n - k \rangle}.$$

**Forward.** If $S$ is independent with $|S| \geq k$, then

$$C = V \setminus S \text{ covers } G, \qquad |C| = n - |S| \leq n - k.$$

**Reverse.** If $C$ covers $G$ with $|C| \leq n - k$, then

$$S = V \setminus C \text{ is independent}, \qquad |S| = n - |C| \geq k.$$

The graph stays the same; the target uses an **at-most** threshold.

# VERTEX-COVER is NP-complete

**Membership in NP.** For an arbitrary input $\langle G, t\rangle$, a certificate lists at most $t$ distinct vertices of $G$.

- Check the vertex identifiers and the size bound.
- For every edge $uv$, check $u \in C$ or $v \in C$.

Using a membership array for $C$, the edge scan takes $O(|V| + |E|)$ time.

**NP-hardness.** We proved

$$\text{INDEPENDENT-SET} \leq_P \text{VERTEX-COVER}, \qquad \langle G, k\rangle \mapsto \langle G, n - k\rangle.$$

Copying $G$ and computing $n - k$ take polynomial time.

Theorem 5.2.1

Since INDEPENDENT-SET is NP-complete, VERTEX-COVER is NP-complete.

# The reduction chain

$$\text{CLIQUE} \leq_P \text{INDEPENDENT-SET} \leq_P \text{VERTEX-COVER}.$$

| Reduction | Graph transformation | Threshold |
|---|---|---|
| Clique to independent set | $G \mapsto \overline{G}$ | $k \mapsto k$ |
| Independent set to cover | $G \mapsto G$ | $k \mapsto n - k$ |

**Checkpoint.** If $G$ has 9 vertices, which vertex-cover instance answers whether $G$ has an independent set of size at least 6?

# Practice: algorithms and reductions

1. Which two implication edges come from $(\neg p \vee q)$?
2. What does it mean if $p$ and $\neg p$ lie in the same SCC?
3. To prove a new problem $B$ NP-hard using CLIQUE, which reduction direction is required? What else proves NP-completeness?
4. A graph $G$ has 8 vertices. Compose today's reductions on the clique instance $\langle G, 5 \rangle$.

# Practice answers

1. $p \to q$ and $\neg q \to \neg p$.
2. The formula is unsatisfiable: either truth value propagates to its opposite.
3. Prove $\mathrm{CLIQUE} \leq_P B$. Also give a polynomial-time verifier to establish $B \in \mathrm{NP}$.
4. Output $\boxed{\langle \overline{G}, 3 \rangle}$: complement the graph, then replace 5 by $8 - 5$.

# Takeaways

- **2SAT is in $\mathrm{P}$.** A formula is satisfiable exactly when no literal shares an SCC with its negation. A topological order gives an assignment.
- **NP-complete means membership plus hardness.** A reduction from a known hard problem transfers hardness to the target.
- $\mathrm{CLIQUE} \leq_P \mathrm{INDEPENDENT\text{-}SET} \leq_P \mathrm{VERTEX\text{-}COVER}$: first complement the graph, then complement the selected vertex set.
- Cook–Levin gives SAT NP-complete. The reductions $\mathrm{SAT} \leq_P 3\mathrm{SAT} \leq_P \mathrm{CLIQUE}$ supply our known hard starting point.