CS 301

Lecture 25 – NP-complete

Stephen Checkoway

April 29, 2018

Polynomial-time reducibility

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial-time computable function if some poly-time TM M exists that halts with just $f(w)$ on its tape when run on w

I.e., f is a computable function as defined before and its TM runs in time $\mathrm{poly}(|w|)$

Polynomial-time reducibility

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial-time computable function if some poly-time TM M exists that halts with just $f(w)$ on its tape when run on w

I.e., f is a computable function as defined before and its TM runs in time $\mathrm{poly}(|w|)$

A is polynomial-time reducible to B (written $A \le_{\mathrm{p}} B$) if a polynomial-time computable function f exists such that $w \in A \iff f(w) \in B$

I.e., $A \le_{\mathrm{m}} B$ and the computable mapping is polynomial time

Another "good-news" reduction theorem

Theorem

If $A \leq_{\mathrm{p}} B$ *and* $B \in \mathrm{P}$*, then* $A \in \mathrm{P}$

Similar proof to all of the others

Another "good-news" reduction theorem

Theorem

If $A \leq_{\mathrm{p}} B$ *and* $B \in \mathrm{P}$, *then* $A \in \mathrm{P}$

Similar proof to all of the others

Proof.

Let f be the poly-time reduction and let M decide B in poly time

M' = "On input w,

1. Compute $f(w)$
2. Run M on $f(w)$; if M accepts, then *accept*; otherwise *reject*"

Another "good-news" reduction theorem

Theorem

If $A \le_{\mathrm{p}} B$ *and* $B \in \mathrm{P}$*, then* $A \in \mathrm{P}$

Similar proof to all of the others

Proof.

Let f be the poly-time reduction and let M decide B in poly time

M' = "On input w,

1. Compute $f(w)$
2. Run M on $f(w)$; if M accepts, then *accept*; otherwise *reject*"

Computing $f(w)$ takes time $\mathrm{poly}(|w|)$ and $|f(w)| = \mathrm{poly}(|w|)$

Running M on $f(w)$ takes time $\mathrm{poly}(|f(w)|) = \mathrm{poly}(\mathrm{poly}(|w|)) = \mathrm{poly}(|w|)$

Thus, M' decides A in polynomial time so $A \in \mathrm{P}$ $\square$

Another "good-news" reduction theorem

Theorem

If $A \leq_{\mathrm{p}} B$ *and* $B \in \mathrm{P}$, *then* $A \in \mathrm{P}$

Similar proof to all of the others

Proof.

Let f be the poly-time reduction and let M decide B in poly time

M' = "On input w,

1. Compute $f(w)$
2. Run M on $f(w)$; if M accepts, then *accept*; otherwise *reject*"

Computing $f(w)$ takes time $\mathrm{poly}(|w|)$ and $|f(w)| = \mathrm{poly}(|w|)$

Running M on $f(w)$ takes time $\mathrm{poly}(|f(w)|) = \mathrm{poly}(\mathrm{poly}(|w|)) = \mathrm{poly}(|w|)$

Thus, M' decides A in polynomial time so $A \in \mathrm{P}$ $\square$

Replacing P with NP in the proof and using NTMs rather than TMs shows that $A \leq_{\mathrm{p}} B$ and $B \in \mathrm{NP}$, then $A \in \mathrm{NP}$

CNF-SAT $\leq_p$ 3-SAT

$$\text{CNF-SAT} = \{\langle\phi\rangle \mid \phi \text{ is in CNF}\}$$
$$\text{3-SAT} = \{\langle\phi\rangle \mid \phi \text{ is in 3-CNF}\}$$

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

To do this, we need to give a poly-time algorithm that converts ϕ in CNF to ϕ' in CNF where each clause has exactly 3 literals

$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where each C_i is a disjunction (OR) of literals

We just need a procedure to convert a clause $C = (u_1 \vee u_2 \vee \cdots \vee u_k)$ to 3-CNF

Converting a clause to 3-CNF

Four cases

1. $C = u$: Output $u_1 \vee u_1 \vee u_1$

Converting a clause to 3-CNF

Four cases

1. $C = u$: Output $u_1 \vee u_1 \vee u_1$
2. $C = u_1 \vee u_2$: Output $u_1 \vee u_2 \vee u_2$

Converting a clause to 3-CNF

Four cases

1. $C = u$: Output $u_1 \vee u_1 \vee u_1$
2. $C = u_1 \vee u_2$: Output $u_1 \vee u_2 \vee u_2$
3. $C = u_1 \vee u_2 \vee u_3$: Output C

Converting a clause to 3-CNF

Four cases

1. $C = u$: Output $u_1 \vee u_1 \vee u_1$
2. $C = u_1 \vee u_2$: Output $u_1 \vee u_2 \vee u_2$
3. $C = u_1 \vee u_2 \vee u_3$: Output C
4. $C = u_1 \vee u_2 \vee \cdots \vee u_k$: Introduce $k-3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$(u_1 \vee u_2 \vee z_2) \wedge \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \vee u_i \vee z_i) \right) \wedge (\overline{z_{k-2}} \vee u_{k-1} \vee u_k)$$

For example,

$$(a \vee b \vee c \vee d \vee e) \mapsto (a \vee b \vee z_2) \wedge (\overline{z_2} \vee c \vee z_3) \wedge (\overline{z_3} \vee d \vee e)$$

Converting a clause to 3-CNF

Four cases

1. $C = u$: Output $u_1 \vee u_1 \vee u_1$
2. $C = u_1 \vee u_2$: Output $u_1 \vee u_2 \vee u_2$
3. $C = u_1 \vee u_2 \vee u_3$: Output C
4. $C = u_1 \vee u_2 \vee \cdots \vee u_k$: Introduce $k-3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$(u_1 \vee u_2 \vee z_2) \wedge \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \vee u_i \vee z_i) \right) \wedge (\overline{z_{k-2}} \vee u_{k-1} \vee u_k)$$

For example,

$$(a \vee b \vee c \vee d \vee e) \mapsto (a \vee b \vee z_2) \wedge (\overline{z_2} \vee c \vee z_3) \wedge (\overline{z_3} \vee d \vee e)$$

Cases 1–3 clearly preserve the property that any assignment that makes C true makes the output true and vice versa

Correctness of case 4

4. $C = u_1 \lor u_2 \lor \cdots \lor u_k$: Introduce $k - 3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$\phi' = (u_1 \lor u_2 \lor z_2) \land \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \lor u_i \lor z_i) \right) \land (\overline{z_{k-2}} \lor u_{k-1} \lor u_k)$$

If $C = T$, then there is some true literal, say $u_i = T$, then $\phi' = T$ by setting

$$z_j = \begin{cases} T & \text{for } j < i \\ F & \text{for } j \geq i \end{cases}$$

Even if all of the other literals are false, setting z_j this way satisfies each clause in ϕ'

Correctness of case 4

4. $C = u_1 \vee u_2 \vee \cdots \vee u_k$: Introduce $k-3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$\phi' = (u_1 \vee u_2 \vee z_2) \wedge \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \vee u_i \vee z_i) \right) \wedge (\overline{z_{k-2}} \vee u_{k-1} \vee u_k)$$

If $C = T$, then there is some true literal, say $u_i = T$, then $\phi' = T$ by setting

$$z_j = \begin{cases} T & \text{for } j < i \\ F & \text{for } j \geq i \end{cases}$$

Even if all of the other literals are false, setting z_j this way satisfies each clause in ϕ'

If $u_1 = u_2 = \cdots = u_k = F$, then no matter how we set the z_j, at least one of the clauses in ϕ' is false:

- For $(u_1 \vee u_2 \vee z_2) = T$, we'd need $z_2 = T$

Correctness of case 4

4. $C = u_1 \vee u_2 \vee \cdots \vee u_k$: Introduce $k-3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$\phi' = (u_1 \vee u_2 \vee z_2) \wedge \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \vee u_i \vee z_i) \right) \wedge (\overline{z_{k-2}} \vee u_{k-1} \vee u_k)$$

If $C = T$, then there is some true literal, say $u_i = T$, then $\phi' = T$ by setting

$$z_j = \begin{cases} T & \text{for } j < i \\ F & \text{for } j \geq i \end{cases}$$

Even if all of the other literals are false, setting z_j this way satisfies each clause in ϕ'

If $u_1 = u_2 = \cdots = u_k = F$, then no matter how we set the z_j, at least one of the clauses in ϕ' is false:

- For $(u_1 \vee u_2 \vee z_2) = T$, we'd need $z_2 = T$
- For $(\overline{z_2} \vee u_3 \vee z_3) = T$, we'd need $z_3 = T$ and so on; thus $z_j = T$ for all $2 \leq j \leq k-2$

Correctness of case 4

4. $C = u_1 \vee u_2 \vee \cdots \vee u_k$: Introduce $k-3$ new variables $z_2, z_3, \cdots z_{k-2}$ and output

$$\phi' = (u_1 \vee u_2 \vee z_2) \wedge \left(\bigwedge_{i=3}^{k-2} (\overline{z_{i-1}} \vee u_i \vee z_i) \right) \wedge (\overline{z_{k-2}} \vee u_{k-1} \vee u_k)$$

If $C = T$, then there is some true literal, say $u_i = T$, then $\phi' = T$ by setting

$$z_j = \begin{cases} T & \text{for } j < i \\ F & \text{for } j \geq i \end{cases}$$

Even if all of the other literals are false, setting z_j this way satisfies each clause in ϕ'

If $u_1 = u_2 = \cdots = u_k = F$, then no matter how we set the z_j, at least one of the clauses in ϕ' is false:

- For $(u_1 \vee u_2 \vee z_2) = T$, we'd need $z_2 = T$
- For $(\overline{z_2} \vee u_3 \vee z_3) = T$, we'd need $z_3 = T$ and so on; thus $z_j = T$ for all $2 \leq j \leq k-2$
- But then $(\overline{z_{k-2}} \vee u_{k-1} \vee u_k) = F$

Proof that $\mathrm{CNF\text{-}SAT} \leq_{\mathrm{p}} \mathrm{3\text{-}SAT}$

Proof.

Define TM T = "On input $\langle \phi \rangle$,

1. For each clause C in ϕ,
2. Convert C to 3-CNF using the given algorithm
3. Output the conjunction (AND) of the result for each clause"

Proof that $\mathrm{CNF\text{-}SAT} \leq_{\mathrm{p}} \mathrm{3\text{-}SAT}$

Proof.

Define TM T = "On input $\langle \phi \rangle$,

1. For each clause C in ϕ,
2. Convert C to 3-CNF using the given algorithm
3. Output the conjunction (AND) of the result for each clause"

If $\langle \phi \rangle \in \mathrm{CNF\text{-}SAT}$, then there is some assignment of truth values to variables that makes $\phi = T$. By setting the extra variables from the algorithm appropriately, the output is satisfiable so $f(\langle \phi \rangle) \in \mathrm{3\text{-}SAT}$

Proof that $\mathrm{CNF\text{-}SAT} \le_{\mathrm{p}} \mathrm{3\text{-}SAT}$

Proof.

Define TM T = "On input $\langle \phi \rangle$,

1. For each clause C in ϕ,
2.   Convert C to 3-CNF using the given algorithm
3. Output the conjunction (AND) of the result for each clause"

If $\langle \phi \rangle \in \mathrm{CNF\text{-}SAT}$, then there is some assignment of truth values to variables that makes $\phi = T$. By setting the extra variables from the algorithm appropriately, the output is satisfiable so $f(\langle \phi \rangle) \in \mathrm{3\text{-}SAT}$

If $\langle \phi \rangle \notin \mathrm{CNF\text{-}SAT}$, then for any assignment, some clause in ϕ is false and by construction, no matter how the extra variables are set for the corresponding output clauses, one of them is false so $f(\langle \phi \rangle) \notin \mathrm{3\text{-}SAT}$

Proof that $\mathrm{CNF\text{-}SAT} \leq_{\mathrm{p}} \mathrm{3\text{-}SAT}$

Proof.

Define TM T = "On input $\langle \phi \rangle$,

1. For each clause C in ϕ,
2. Convert C to 3-CNF using the given algorithm
3. Output the conjunction (AND) of the result for each clause"

If $\langle \phi \rangle \in \mathrm{CNF\text{-}SAT}$, then there is some assignment of truth values to variables that makes $\phi = T$. By setting the extra variables from the algorithm appropriately, the output is satisfiable so $f(\langle \phi \rangle) \in \mathrm{3\text{-}SAT}$

If $\langle \phi \rangle \notin \mathrm{CNF\text{-}SAT}$, then for any assignment, some clause in ϕ is false and by construction, no matter how the extra variables are set for the corresponding output clauses, one of them is false so $f(\langle \phi \rangle) \notin \mathrm{3\text{-}SAT}$

If ϕ has n total literals, then the output of T has less than $3n$ clauses each of which has 3 literals so $|f(\langle \phi \rangle)| = \mathrm{poly}(|\langle \phi \rangle|)$ thus T takes polynomial time □

NP-hard and NP-complete

Language B is NP-hard if every language $A \in \mathrm{NP}$ is poly-time reducible to B $(\forall A \in \mathrm{NP}.\ A \leq_{\mathrm{p}} B)$

NP-hard and NP-complete

Language B is NP-hard if every language $A \in \mathrm{NP}$ is poly-time reducible to B
($\forall A \in \mathrm{NP}.\ A \leq_{\mathrm{p}} B$)

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

Equivalently, B is NP-complete if

1. $B \in \mathrm{NP}$
2. $\forall A \in \mathrm{NP}.\ A \leq_{\mathrm{p}} B$

NP-hard and NP-complete

Language B is NP-hard if every language $A \in \mathrm{NP}$ is poly-time reducible to B ($\forall A \in \mathrm{NP}.\ A \leq_{\mathrm{p}} B$)

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

Equivalently, B is NP-complete if

1. $B \in \mathrm{NP}$
2. $\forall A \in \mathrm{NP}.\ A \leq_{\mathrm{p}} B$

NP-complete problems represent the "hardest" problems in NP to solve

Any efficient solution to an NP-complete problem gives an efficient solution to every problem in NP

P, NP, and NP-complete

Theorem

If B *is* NP*-complete and* $B \in \mathrm{P}$*, then* $\mathrm{P} = \mathrm{NP}$

How would we prove this?

P, NP, and NP-complete

Theorem

If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$

How would we prove this?

Proof.

If $A \in \mathrm{NP}$, then $A \leq_{\mathrm{p}} B$ and since $B \in \mathrm{P}$, $A \in \mathrm{P}$ $\square$

P, NP, and NP-complete

Theorem

If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$

How would we prove this?

Proof.

If $A \in \mathrm{NP}$, then $A \leq_{\mathrm{p}} B$ and since $B \in \mathrm{P}$, $A \in \mathrm{P}$ $\square$

This gives us a way to try to prove that $\mathrm{P} = \mathrm{NP}$: Find an NP-complete problem and give a polynomial-time solution

Poly-time reductions between NP-complete problems

Theorem

If B is NP-complete, $C \in \mathrm{NP}$, and $B \leq_{\mathrm{p}} C$, then C is NP-complete

Poly-time reductions between NP-complete problems

Theorem

If B is NP-complete, $C \in \mathrm{NP}$, and $B \leq_{\mathrm{p}} C$, then C is NP-complete

Proof.

Let $A \in \mathrm{NP}$. Because B is NP-complete, $A \leq_{\mathrm{p}} B$

Poly-time reduction is transitive and $B \leq_{\mathrm{p}} C$ so $A \leq_{\mathrm{p}} C$ thus C is NP-hard and because $C \in \mathrm{NP}$, C is NP-complete $\square$

Poly-time reductions between NP-complete problems

Theorem

If B is NP-complete, $C \in \mathrm{NP}$, and $B \leq_{\mathrm{p}} C$, then C is NP-complete

Proof.

Let $A \in \mathrm{NP}$. Because B is NP-complete, $A \leq_{\mathrm{p}} B$

Poly-time reduction is transitive and $B \leq_{\mathrm{p}} C$ so $A \leq_{\mathrm{p}} C$ thus C is NP-hard and because $C \in \mathrm{NP}$, C is NP-complete $\square$

Once we have one NP-complete problem, this gives us a way to find a bunch more, but we need to find one to start us off

Cook–Levin theorem

Theorem

SAT is NP-complete

Cook–Levin theorem

Theorem

SAT is NP-complete

Sipser's proof actually shows that $\mathrm{CNF\text{-}SAT}$ is NP-complete

We showed that $\mathrm{SAT} \in \mathrm{NP}$ and a boolean formula in CNF is, of course, a boolean formula so $\langle \phi \rangle \mapsto \langle \phi \rangle$ is polynomial-time reduction from $\mathrm{CNF\text{-}SAT}$ to SAT

3-SAT is NP-complete

Theorem

$\mathit{3\text{-}SAT}$ is NP-complete

To prove this, we need to show two things: $\text{3-SAT} \in \text{NP}$ and there is some NP-complete language A that poly-time reduces to 3-SAT

3-SAT is NP-complete

Theorem

3-SAT is NP-complete

To prove this, we need to show two things: $\mathrm{3\text{-}SAT} \in \mathrm{NP}$ and there is some NP-complete language A that poly-time reduces to $\mathrm{3\text{-}SAT}$

Proof.

We already showed that $\mathrm{CNF\text{-}SAT} \le_{\mathrm{p}} \mathrm{3\text{-}SAT}$ so all that remains is to show that $\mathrm{3\text{-}SAT} \in \mathrm{NP}$

But this is true for the same reason $\mathrm{SAT} \in \mathrm{NP}$: We can verify an assignment of truth values to variables satisfies a formula in poly time $\square$

General technique

If we want to show that some language L is NP-complete, then we need to perform 3 steps

1. Show that $L \in \mathrm{NP}$
2. Select some NP-complete language B
3. Show that $B \leq_{\mathrm{p}} L$

Step 1 is frequently easy: If the language is of the form $\{w \mid w \text{ has some property or structure}\}$, then the certificate for the verifier is whatever makes the property true or the structure itself

Steps 2 and 3 can be quite challenging and can require a great deal of cleverness; $3\text{-}\mathrm{SAT}$ is usually a good first choice for the NP-complete language

VertexCover is NP-complete

Recall VertexCover = $\{\langle G, k\rangle \mid G$ has a vertex cover of size $k\} \in \mathrm{NP}$ because the vertex cover itself is the certificate

To show that VertexCover is NP-complete, we want to give a poly-time reduction from 3-SAT

$\textsc{VertexCover}$ is NP-complete

Recall $\textsc{VertexCover} = \{\langle G, k\rangle \mid G$ has a vertex cover of size $k\} \in \mathrm{NP}$ because the vertex cover itself is the certificate

To show that $\textsc{VertexCover}$ is NP-complete, we want to give a poly-time reduction from 3-SAT

To do this, we want to take a formula ϕ that has m clauses $C_1, C_2, \ldots, C_m$ and n variables $x_1, x_2, \ldots, x_n$ and construct an undirected graph $G = (V, E)$ and a k s.t. G has a vertex cover of size k iff ϕ is satisfiable

That is, we need to produce a mapping $\langle\phi\rangle \mapsto \langle G, k\rangle$ such that $\langle\phi\rangle \in 3\text{-SAT} \iff \langle G, k\rangle \in \textsc{VertexCover}$ and we have to be able to compute the mapping in some polynomial of m and n

Gadgets

For each variable and each clause, we want to construct some portion of a graph

Running example: $\phi = (\underbrace{x_1 \vee \overline{x_2} \vee x_3}_{C_1}) \wedge (\underbrace{\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}_{C_2})$

Gadgets

For each variable and each clause, we want to construct some portion of a graph

Running example: $\phi = (\underbrace{x_1 \vee \overline{x_2} \vee x_3}_{C_1}) \wedge (\underbrace{\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}_{C_2})$

1. **Assignment.** For each variable x_i create vertices x_i and $\overline{x_i}$ and edge $(x_i, \overline{x_i})$

$$V_A = \bigcup_{i=1}^{n} \{x_i, \overline{x_i}\}$$

$$E_A = \bigcup_{i=1}^{n} \{(x_i, \overline{x_i})\}$$

x_1 —— $\overline{x_1}$ $\quad$ x_2 —— $\overline{x_2}$ $\quad$ x_3 —— $\overline{x_3}$

Gadgets

For each variable and each clause, we want to construct some portion of a graph

Running example: $\phi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{C_1} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})}_{C_2}$

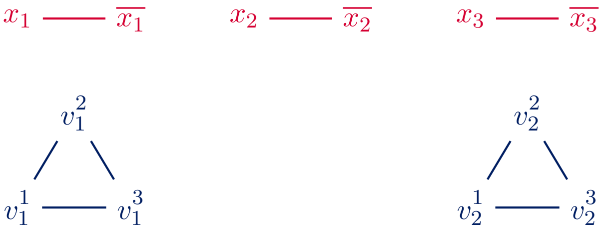
1. **Assignment.** For each variable x_i create vertices x_i and $\overline{x_i}$ and edge $(x_i, \overline{x_i})$
2. **Satisfiability.** For each clause $C_j = (a_j \vee b_j \vee c_j)$, create vertices v_j^1, v_j^2, and v_j^3 with edges between them

$$V_A = \bigcup_{i=1}^{n} \{x_i, \overline{x_i}\}$$

$$E_A = \bigcup_{i=1}^{n} \{(x_i, \overline{x_i})\}$$

$$V_S = \bigcup_{j=1}^{m} \{v_j^1, v_j^2, v_j^3\}$$

$$E_S = \bigcup_{j=1}^{m} \{(v_j^1, v_j^2), (v_j^2, v_j^3), (v_j^3, v_j^1)\}$$

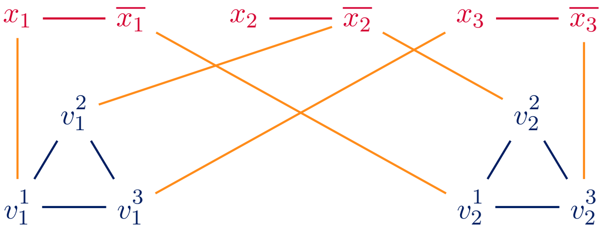


Gadgets

For each variable and each clause, we want to construct some portion of a graph

Running example: $\phi = (\underbrace{x_1 \vee \overline{x_2} \vee x_3}_{C_1}) \wedge (\underbrace{\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}_{C_2})$

1. **Assignment.** For each variable x_i create vertices x_i and $\overline{x_i}$ and edge $(x_i, \overline{x_i})$
2. **Satisfiability.** For each clause $C_j = (a_j \vee b_j \vee c_j)$, create vertices v_j^1, v_j^2, and v_j^3 with edges between them
3. **Communication.** For each clause $C_j = (a_j \vee b_j \vee c_j)$, create edges (v_j^1, a_j), (v_j^2, b_j), and (v_j^3, c_j)



$$V_A = \bigcup_{i=1}^{n} \{x_i, \overline{x_i}\}$$

$$E_A = \bigcup_{i=1}^{n} \{(x_i, \overline{x_i})\}$$

$$V_S = \bigcup_{j=1}^{m} \{v_j^1, v_j^2, v_j^3\}$$

$$E_S = \bigcup_{j=1}^{m} \{(v_j^1, v_j^2), (v_j^2, v_j^3), (v_j^3, v_j^1)\}$$

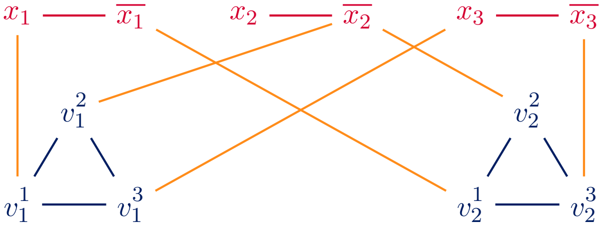
$$E_C = \bigcup_{j=1}^{m} \{(v_j^1, a_j), (v_j^2, b_j), (v_j^3, c_j)\}$$

Gadgets

For each variable and each clause, we want to construct some portion of a graph

Running example: $\phi = (\underbrace{x_1 \lor \overline{x_2} \lor x_3}_{C_1}) \land (\underbrace{\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}}_{C_2})$

1. **Assignment.** For each variable x_i create vertices x_i and $\overline{x_i}$ and edge $(x_i, \overline{x_i})$
2. **Satisfiability.** For each clause $C_j = (a_j \lor b_j \lor c_j)$, create vertices v_j^1, v_j^2, and v_j^3 with edges between them
3. **Communication.** For each clause $C_j = (a_j \lor b_j \lor c_j)$, create edges (v_j^1, a_j), (v_j^2, b_j), and (v_j^3, c_j)



$$V_A = \bigcup_{i=1}^{n} \{x_i, \overline{x_i}\}$$

$$E_A = \bigcup_{i=1}^{n} \{(x_i, \overline{x_i})\}$$

$$V_S = \bigcup_{j=1}^{m} \{v_j^1, v_j^2, v_j^3\}$$

$$E_S = \bigcup_{j=1}^{m} \{(v_j^1, v_j^2), (v_j^2, v_j^3), (v_j^3, v_j^1)\}$$

$$E_C = \bigcup_{j=1}^{m} \{(v_j^1, a_j), (v_j^2, b_j), (v_j^3, c_j)\}$$

Output: $G = (V, E)$, k where

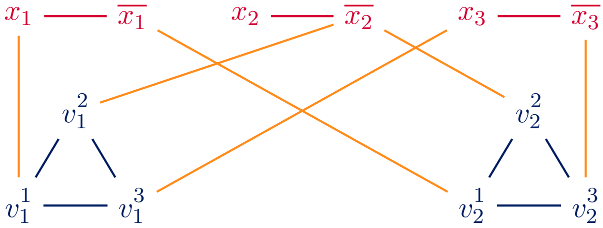
$V = V_A \cup V_S$

$E = E_A \cup E_S \cup E_C$

$k = n + 2m$

If G has a VC of size $n + 2m$...

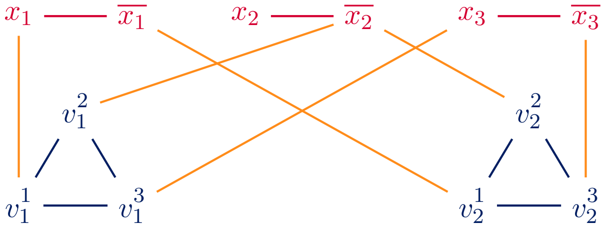
$$\phi = (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3})$$



If G has a vertex cover VC of size $n + 2m$, then to cover the n assignment edges, at least n of the literal vertices must be in VC

If G has a VC of size $n + 2m$...

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

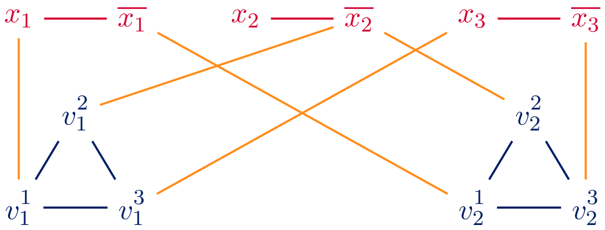


If G has a vertex cover VC of size $n + 2m$, then to cover the n assignment edges, at least n of the literal vertices must be in VC

To cover the satisfiability edges, at least 2 vertices in each triangle must be in VC

If G has a VC of size $n + 2m$...

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

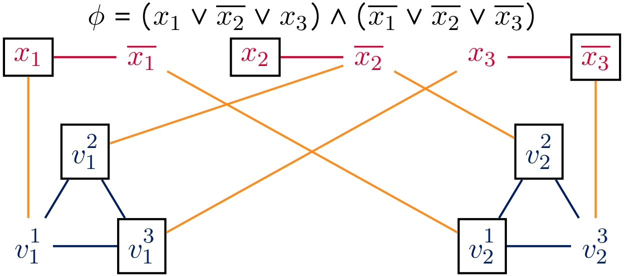


If G has a vertex cover VC of size $n + 2m$, then to cover the n assignment edges, at least n of the literal vertices must be in VC

To cover the satisfiability edges, at least 2 vertices in each triangle must be in VC

Thus VC contains exactly n of the assignment vertices, either x_i or $\overline{x_i}$ for each i and exactly 2 of each of the m satisfiability triangles

If G has a VC of size $n + 2m$...



If G has a vertex cover VC of size $n + 2m$, then to cover the n assignment edges, at least n of the literal vertices must be in VC

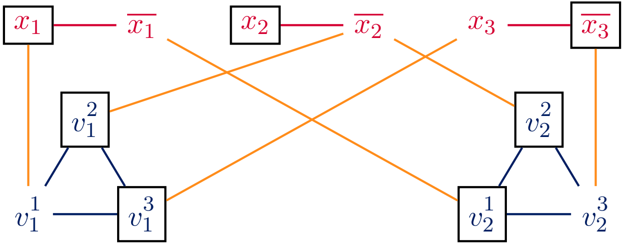
To cover the satisfiability edges, at least 2 vertices in each triangle must be in VC

Thus VC contains exactly n of the assignment vertices, either x_i or $\overline{x_i}$ for each i and exactly 2 of each of the m satisfiability triangles

For example, the boxed vertices form a vertex cover of size $n + 2m = 7$

If G has a VC of size $n + 2m$, then ϕ is satisfiable

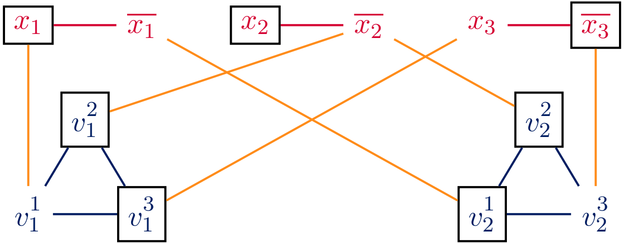
$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



Create a satisfying assignment for ϕ by setting $x_i = T$ if node $x_i \in VC$

If G has a VC of size $n + 2m$, then ϕ is satisfiable

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



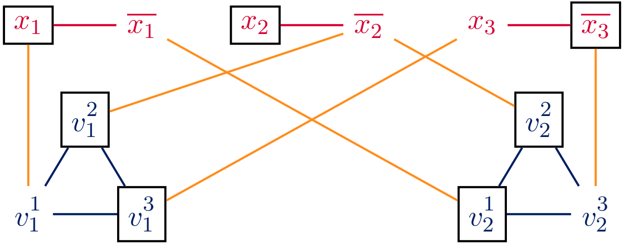
Create a satisfying assignment for ϕ by setting $x_i = T$ if node $x_i \in VC$

Consider the triangle corresponding to clause C_j, 2 of the vertices are in VC and the third is connected to its literal which must be in VC in order to cover the communication edge

For example, edge (v_1^1, x_1) is covered by $x_1 \in VC$ so clause C_1 is satisfied
Similarly for edge $(v_2^3, \overline{x_3})$ and clause C_2

If G has a VC of size $n + 2m$, then ϕ is satisfiable

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



Create a satisfying assignment for ϕ by setting $x_i = T$ if node $x_i \in VC$

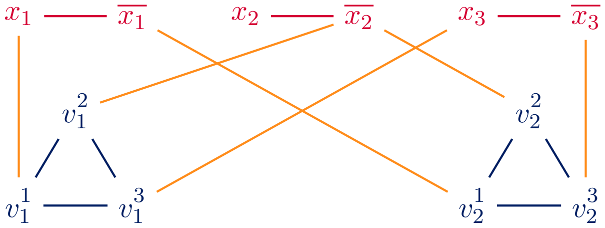
Consider the triangle corresponding to clause C_j, 2 of the vertices are in VC and the third is connected to its literal which must be in VC in order to cover the communication edge

For example, edge (v_1^1, x_1) is covered by $x_1 \in VC$ so clause C_1 is satisfied
Similarly for edge $(v_2^3, \overline{x_3})$ and clause C_2

Thus each clause is satisfied so ϕ is satisfied

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

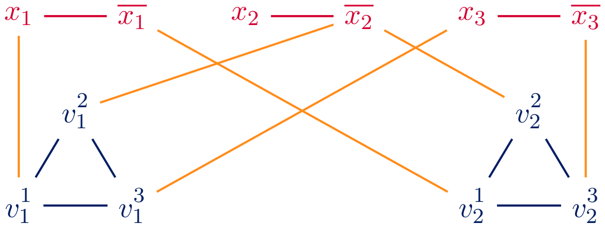
$$\phi = (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3})$$



If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

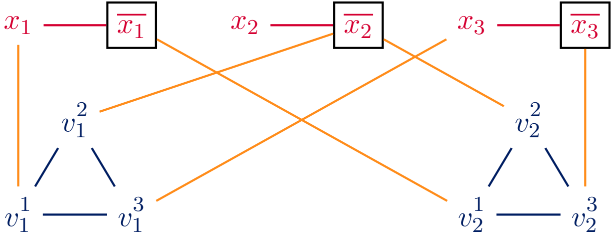


If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3})$$

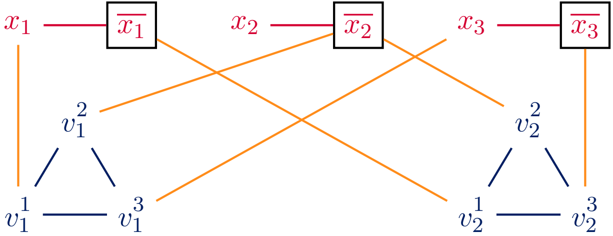


If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



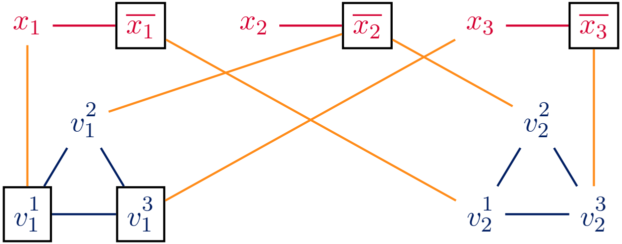
If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

Now (v_1^1, x_1) and (v_1^3, x_3) need to be covered so add v_1^1 and v_1^3 to VC

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



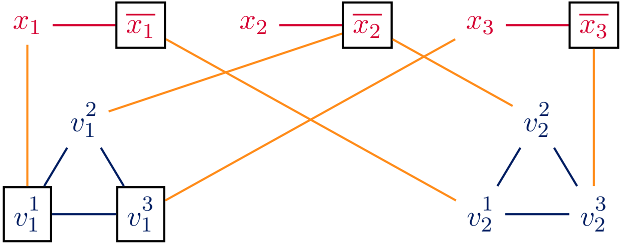
If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

Now (v_1^1, x_1) and (v_1^3, x_3) need to be covered so add v_1^1 and v_1^3 to VC

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

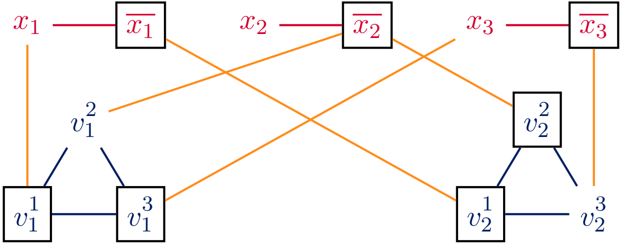
For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

Now (v_1^1, x_1) and (v_1^3, x_3) need to be covered so add v_1^1 and v_1^3 to VC

All of the communication edges for clause C_2 are covered, so pick any 2 vertices

If ϕ is satisfied by some assignment, then G has a VC of size $n + 2m$

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$



If ϕ is satisfied by some assignment, then we can construct a vertex cover of size $n + 2m$ consisting of each of the true assignment literals and two of the satisfiability vertices of each clause as required to cover the communication edges connected to false literals

For example, $x_1 = x_2 = x_3 = F$ satisfies ϕ so first put $\overline{x_1}$, $\overline{x_2}$, and $\overline{x_3}$ in VC

Now (v_1^1, x_1) and (v_1^3, x_3) need to be covered so add v_1^1 and v_1^3 to VC

All of the communication edges for clause C_2 are covered, so pick any 2 vertices

So VertexCover is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

So $\textsc{VertexCover}$ is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$

So VertexCover is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$
2. We gave a construction $\langle \phi \rangle \mapsto \langle G, k \rangle$

So $\textsc{VertexCover}$ is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$
2. We gave a construction $\langle \phi \rangle \mapsto \langle G, k \rangle$
3. We showed that if G has a vertex cover of size k (i.e., $\langle G, k \rangle \in \textsc{VertexCover}$), then ϕ is satisfiable (i.e., $\langle \phi \rangle \in \text{3-SAT}$)

So $\textsc{VertexCover}$ is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$
2. We gave a construction $\langle \phi \rangle \mapsto \langle G, k \rangle$
3. We showed that if G has a vertex cover of size k (i.e., $\langle G, k \rangle \in \textsc{VertexCover}$), then ϕ is satisfiable (i.e., $\langle \phi \rangle \in 3\text{-SAT}$)
4. We showed that if ϕ is satisfiable, then G has a vertex cover of size k

So $\textsc{VertexCover}$ is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$
2. We gave a construction $\langle \phi \rangle \mapsto \langle G, k \rangle$
3. We showed that if G has a vertex cover of size k (i.e., $\langle G, k \rangle \in \textsc{VertexCover}$), then ϕ is satisfiable (i.e., $\langle \phi \rangle \in 3\text{-SAT}$)
4. We showed that if ϕ is satisfiable, then G has a vertex cover of size k
5. We argued that the construction takes polynomial time

So $\textsc{VertexCover}$ is NP-complete

Lastly, G takes polynomial time to create since it has a polynomial number of vertices and edges

Recap

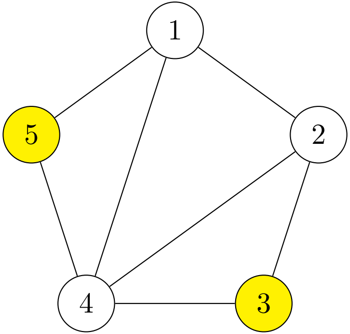
1. We showed that $\textsc{VertexCover} \in \mathrm{NP}$
2. We gave a construction $\langle \phi \rangle \mapsto \langle G, k \rangle$
3. We showed that if G has a vertex cover of size k (i.e., $\langle G, k \rangle \in \textsc{VertexCover}$), then ϕ is satisfiable (i.e., $\langle \phi \rangle \in 3\text{-SAT}$)
4. We showed that if ϕ is satisfiable, then G has a vertex cover of size k
5. We argued that the construction takes polynomial time

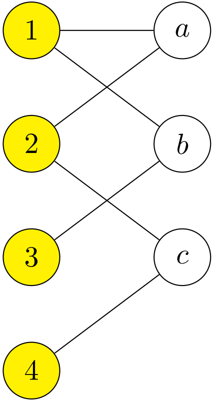
Steps 2–5 show $3\text{-SAT} \le_{\mathrm{p}} \textsc{VertexCover}$ and thus $\textsc{VertexCover}$ is NP-complete

Independent set

If $G = (V, E)$ is an undirected graph, an independent set is a set $I \subseteq V$ such that no two vertices in I are adjacent
I.e., $\forall u, v \in I \ (u, v) \notin E$

E.g., the yellow vertices form an independent set





IndSet

IndSet = $\{\langle G, k\rangle \mid G$ is an undirected graph which has an independent set of size $k\}$

How would we show that IndSet is NP-complete?

IndSet

$\textsc{IndSet} = \{\langle G, k\rangle \mid G$ is an undirected graph which has an independent set of size $k\}$

How would we show that IndSet is NP-complete?

We need to show

1. $\textsc{IndSet} \in \mathrm{NP}$ and
2. There is some A which is NP-complete and $A \le_{\mathrm{p}} \textsc{IndSet}$

IndSet $\in$ NP

What is a certificate for IndSet?

IndSet $\in$ NP

What is a certificate for IndSet?

The independent set I of size k.

IndSet $\in$ NP

What is a certificate for IndSet?

The independent set I of size k.

We can build a verifier for IndSet:

V = "On input $\langle G, k, I\rangle$ where $G = (V, E)$,

1. If $I \nsubseteq V$ or $|I| \neq k$, then *reject*
2. For each $(u, v) \in E$,
3. If $u \in I$ and $v \in I$, then *reject*
4. Otherwise *accept*"

Each step clearly takes polynomial time and the body of the loop happens once per edge so V is a polynomial time verifier

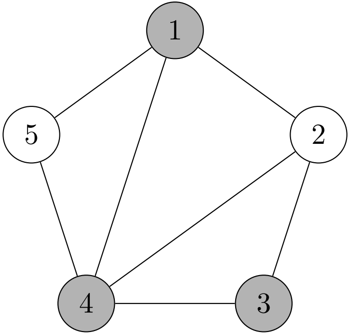
VertexCover $\leq_p$ IndSet

We can reduce from VertexCover to IndSet by giving a polynomial time map $\langle G, k\rangle \mapsto \langle G', k'\rangle$ such that $\langle G, k\rangle \in \text{VertexCover} \iff \langle G', k'\rangle \in \text{IndSet}$

VertexCover $\leq_p$ IndSet

We can reduce from VertexCover to IndSet by giving a polynomial time map $\langle G, k\rangle \mapsto \langle G', k'\rangle$ such that $\langle G, k\rangle \in \text{VertexCover} \iff \langle G', k'\rangle \in \text{IndSet}$

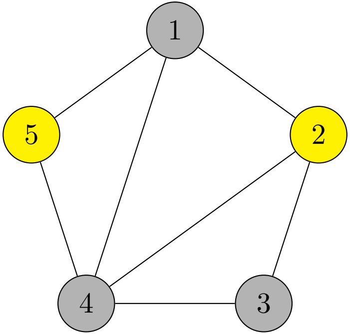
Grey vertices form a vertex cover, What are some independent sets?



VERTEXCOVER $\leq_p$ INDSET

We can reduce from VERTEXCOVER to INDSET by giving a polynomial time map $\langle G, k \rangle \mapsto \langle G', k' \rangle$ such that
$\langle G, k \rangle \in$ VERTEXCOVER $\iff$ $\langle G', k' \rangle \in$ INDSET

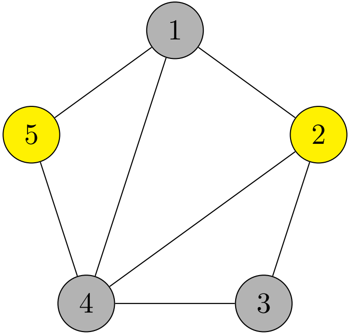
Grey vertices form a vertex cover, What are some independent sets?

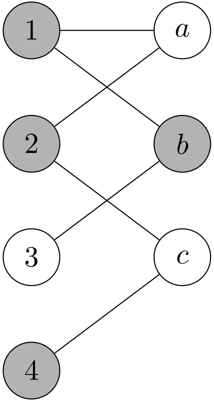


VertexCover $\le_p$ IndSet

We can reduce from VertexCover to IndSet by giving a polynomial time map $\langle G, k\rangle \mapsto \langle G', k'\rangle$ such that
$\langle G, k\rangle \in$ VertexCover $\iff$ $\langle G', k'\rangle \in$ IndSet

Grey vertices form a vertex cover, What are some independent sets?

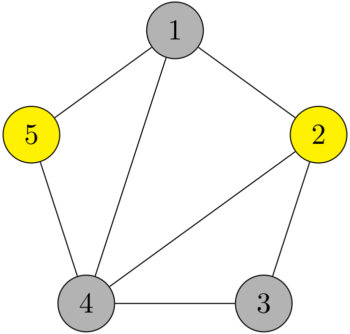


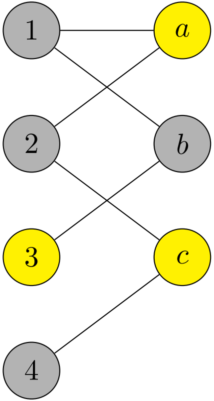


VertexCover $\leq_p$ IndSet

We can reduce from VertexCover to IndSet by giving a polynomial time map $\langle G, k \rangle \mapsto \langle G', k' \rangle$ such that
$\langle G, k \rangle \in$ VertexCover $\iff \langle G', k' \rangle \in$ IndSet

Grey vertices form a vertex cover, What are some independent sets?





Relationship between vertex covers and independent sets

It *looks* like if $G = (V, E)$ has a vertex cover C, then $I = V \setminus C$ is an independent set, and vice versa

Can we prove that?

Relationship between vertex covers and independent sets

It *looks* like if $G = (V, E)$ has a vertex cover C, then $I = V \setminus C$ is an independent set, and vice versa

Can we prove that?

Yes!

Relationship between vertex covers and independent sets

It *looks* like if $G = (V, E)$ has a vertex cover C, then $I = V \setminus C$ is an independent set, and vice versa

Can we prove that?

Yes!

- If $C \subseteq V$ is a vertex cover for G and $I = V \setminus C$, then for all $(u, v) \in E$, either $u \in C$ or $v \in C$. Therefore, for all $u, v \in I$, $(u, v) \notin E$ so I is an independent set

Relationship between vertex covers and independent sets

It *looks* like if $G = (V, E)$ has a vertex cover C, then $I = V \setminus C$ is an independent set, and vice versa

Can we prove that?

Yes!

- If $C \subseteq V$ is a vertex cover for G and $I = V \setminus C$, then for all $(u, v) \in E$, either $u \in C$ or $v \in C$. Therefore, for all $u, v \in I$, $(u, v) \notin E$ so I is an independent set
- If $I \subseteq V$ is an independent set and $C = V \setminus I$, then for all $(u, v) \in E$, at least one of u or v is in C [why?] so C is a vertex cover

How does this help us?

Relationship between vertex covers and independent sets

It *looks* like if $G = (V, E)$ has a vertex cover C, then $I = V \setminus C$ is an independent set, and vice versa

Can we prove that?

Yes!

- If $C \subseteq V$ is a vertex cover for G and $I = V \setminus C$, then for all $(u, v) \in E$, either $u \in C$ or $v \in C$. Therefore, for all $u, v \in I$, $(u, v) \notin E$ so I is an independent set
- If $I \subseteq V$ is an independent set and $C = V \setminus I$, then for all $(u, v) \in E$, at least one of u or v is in C [why?] so C is a vertex cover

How does this help us?

It means that G has n vertices, then G has a vertex cover of size k iff G has an independent set of size $n - k$

VertexCover $\leq_p$ IndSet

Proof.

Our polynomial time mapping is $\langle G, k \rangle \mapsto \langle G, n - k \rangle$ where $G = (V, E)$ and $|V| = n$

$\textsc{VertexCover} \le_p \textsc{IndSet}$

Proof.

Our polynomial time mapping is $\langle G, k\rangle \mapsto \langle G, n-k\rangle$ where $G = (V, E)$ and $|V| = n$

Since G has a vertex cover of size k iff G has an independent set of size $n - k$,

$$\langle G, k\rangle \in \textsc{VertexCover} \iff \langle G, n-k\rangle \in \textsc{IndSet}$$

Since $\textsc{IndSet} \in \mathrm{NP}$, $\textsc{VertexCover} \le_p \textsc{IndSet}$, and $\textsc{VertexCover}$ is NP-complete, $\textsc{IndSet}$ is NP-complete $\square$

CLIQUE is NP-complete

We already proved that CLIQUE $\in$ NP so all that remains is to give a polynomial time mapping from some NP-complete problem

Let's use INDSET

$\textsc{Clique}$ is NP-complete

We already proved that $\textsc{Clique} \in \mathrm{NP}$ so all that remains is to give a polynomial time mapping from some NP-complete problem

Let's use $\textsc{IndSet}$

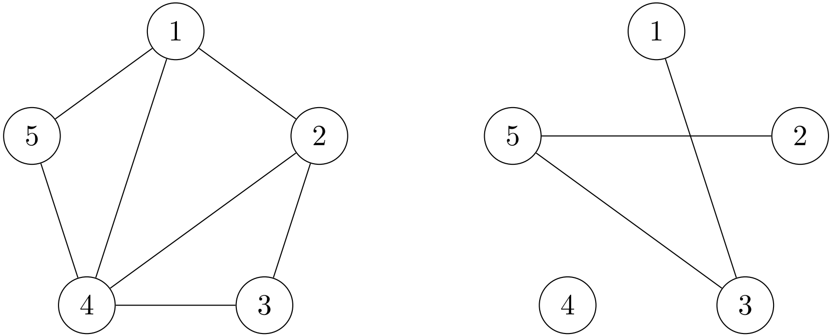
We want a mapping $\langle G, k\rangle \mapsto \langle G', k'\rangle$ such that G has an independent set of size k iff G' has a clique of size k'

Recall

- **Independent set.** I is an independent set if there is no edge between any two vertices in I
- **Clique.** C is a clique if there is an edge between every two vertices in C

Complement of a graph

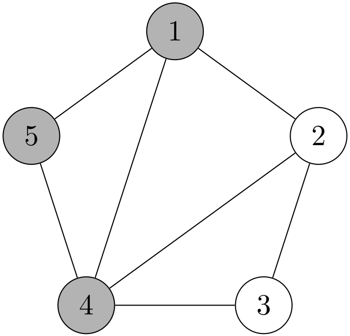
The complement of a graph $G = (V, E)$ is a graph $G' = (V, E')$ where $(u, v) \in E$ iff $(u, v) \notin E'$ (assuming $u \neq v$)

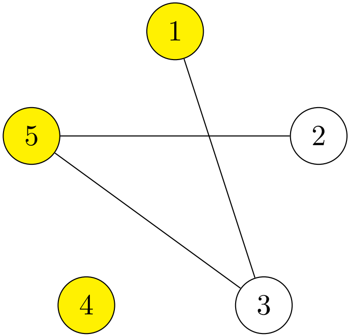


Complement of a graph

The complement of a graph $G = (V, E)$ is a graph $G' = (V, E')$ where $(u, v) \in E$ iff $(u, v) \notin E'$ (assuming $u \neq v$)

Grey vertices form a clique, yellow form an independent set

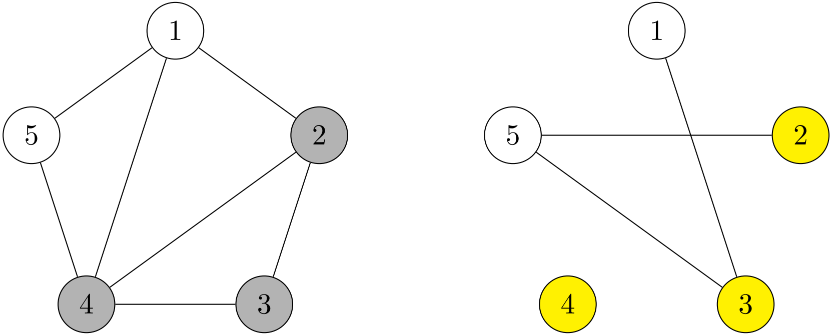




Complement of a graph

The complement of a graph $G = (V, E)$ is a graph $G' = (V, E')$ where $(u, v) \in E$ iff $(u, v) \notin E'$ (assuming $u \neq v$)

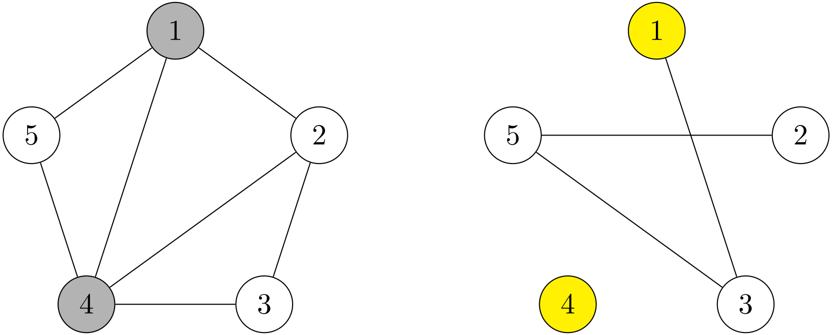
Grey vertices form a clique, yellow form an independent set



Complement of a graph

The complement of a graph $G = (V, E)$ is a graph $G' = (V, E')$ where $(u, v) \in E$ iff $(u, v) \notin E'$ (assuming $u \neq v$)

Grey vertices form a clique, yellow form an independent set



Relationship between a clique and an independent set

Again, this suggests a relationship between cliques and independent sets that we can prove

Let $G = (V, E)$ be an undirected graph and $G' = (V, E')$ be its complement

Relationship between a clique and an independent set

Again, this suggests a relationship between cliques and independent sets that we can prove

Let $G = (V, E)$ be an undirected graph and $G' = (V, E')$ be its complement

- If $C \subseteq V$ is a clique in G, then for each distinct $u, v \in C$, $(u, v) \in E$ and thus $(u, v) \notin E'$ so C is an independent set in G'

Relationship between a clique and an independent set

Again, this suggests a relationship between cliques and independent sets that we can prove

Let $G = (V, E)$ be an undirected graph and $G' = (V, E')$ be its complement

- If $C \subseteq V$ is a clique in G, then for each distinct $u, v \in C$, $(u, v) \in E$ and thus $(u, v) \notin E'$ so C is an independent set in G'
- And vice versa

IndSet $\leq_p$ Clique

The polynomial time mapping is $\langle G, k \rangle \mapsto \langle G', k \rangle$ where G' is the complement of G

Since Clique $\in$ NP and IndSet $\leq_p$ Clique, Clique is NP-complete

Other NP-complete and related problems

- There are *many* other NP-complete problems

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems
- Since then, more NP-complete problems have been found

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems
- Since then, more NP-complete problems have been found
- There are problems known to be in NP which we don't know to be either in P or NP-complete

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems
- Since then, more NP-complete problems have been found
- There are problems known to be in NP which we don't know to be either in P or NP-complete
- NP-intermediate is the class of language that are in NP but are neither in P nor are NP-complete. In 1975, Richard Ladner proved that if $\mathrm{P} \neq \mathrm{NP}$, then there are NP-intermediate problems (if $P = \mathrm{NP}$, then there are none)

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems
- Since then, more NP-complete problems have been found
- There are problems known to be in NP which we don't know to be either in P or NP-complete
- NP-intermediate is the class of language that are in NP but are neither in P nor are NP-complete. In 1975, Richard Ladner proved that if $\mathrm{P} \neq \mathrm{NP}$, then there are NP-intermediate problems (if $P = \mathrm{NP}$, then there are none)
- co-NP is the class of languages whose complements are in NP

Other NP-complete and related problems

- There are *many* other NP-complete problems
- In 1971, Richard Karp gave a list of 21 combinatorial and graph problems that he showed were NP-complete by reducing from SAT
- In 1979, Michael Garey and David Johnson published a (fantastic) book containing a massive list of NP-complete problems organized with nice reductions from earlier problems
- Since then, more NP-complete problems have been found
- There are problems known to be in NP which we don't know to be either in P or NP-complete
- NP-intermediate is the class of language that are in NP but are neither in P nor are NP-complete. In 1975, Richard Ladner proved that if $\mathrm{P} \neq \mathrm{NP}$, then there are NP-intermediate problems (if $P = \mathrm{NP}$, then there are none)
- co-NP is the class of languages whose complements are in NP
- There are languages in NP and co-NP which aren't known to be in P ($\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$)