Computability and Complexity
COSC 4200

NP-Complete Problems

Polynomial-Time Reductions

Definition

We say that A is *polynomial-time mapping reducible* to B, and write $A \leq_{\mathrm{P}} B$, if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$,

$$w \in A \Leftrightarrow f(w) \in B.$$

Polynomial-Time Reductions

Definition

We say that A is *polynomial-time mapping reducible* to B, and write $A \leq_{\mathrm{P}} B$, if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$,

$$w \in A \Leftrightarrow f(w) \in B.$$

Two implications:

$$\begin{aligned} w \in A &\Rightarrow f(w) \in B \\ f(w) \in B &\Rightarrow w \in A \end{aligned}$$

Polynomial-Time Reductions

Definition

We say that A is *polynomial-time mapping reducible* to B, and write $A \leq_{\mathrm{P}} B$, if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$,

$$w \in A \Leftrightarrow f(w) \in B.$$

Two implications:

$$\begin{aligned} w \in A &\Rightarrow f(w) \in B \\ f(w) \in B &\Rightarrow w \in A \end{aligned}$$

Or equivalently:

$$\begin{aligned} w \in A &\Rightarrow f(w) \in B \\ w \notin A &\Rightarrow f(w) \notin B \end{aligned}$$

NP-Completeness

Definition

A problem B is *NP-complete* if

1. B is in NP, and
2. every A in NP is $\leq_{\mathrm{P}}$-reducible to B.

NP-Completeness

Definition

A problem B is *NP-complete* if

1. B is in NP, and
2. every A in NP is $\leq_{\mathrm{P}}$-reducible to B.

Theorem

The following are equivalent:

1. *Some* NP*-complete problem is in* P*.*
2. *Every* NP*-complete problem is in* P*.*
3. $\mathrm{P} = \mathrm{NP}$*.*

NP-Completeness

Theorem

The following are equivalent:

1. *Some* NP*-complete problem is in* P*.*
2. *Every* NP*-complete problem is in* P*.*
3. $\mathrm{P} = \mathrm{NP}$*.*

NP-Completeness

Theorem

The following are equivalent:

1. *Some* NP*-complete problem is in* P*.*
2. *Every* NP*-complete problem is in* P*.*
3. $\mathrm{P} = \mathrm{NP}$*.*

Proof. 3 $\Rightarrow$ 2 and 2 $\Rightarrow$ 1 are trivial.

NP-Completeness

Theorem

The following are equivalent:

1. *Some* NP*-complete problem is in* P*.*
2. *Every* NP*-complete problem is in* P*.*
3. $\mathrm{P} = \mathrm{NP}$*.*

Proof. 3 $\Rightarrow$ 2 and 2 $\Rightarrow$ 1 are trivial.

To see 1 $\Rightarrow$ 3, assume 1 is true and let B be an NP-complete problem with $B \in \mathrm{P}$.

- Let $A \in \mathrm{NP}$ be arbitrary.
- Then $A \leq_{\mathrm{P}} B$ because B is NP-complete.
- Because $B \in \mathrm{P}$, we have $A \in \mathrm{P}$ as well.

Therefore $\mathrm{P} = \mathrm{NP}$. $\square$

NP-Completeness

Theorem

The following are equivalent:

1. *Some* NP*-complete problem is in* P.
2. *Every* NP*-complete problem is in* P.
3. $\mathrm{P} = \mathrm{NP}$.

Corollary

The following are equivalent:

1. *Some* NP*-complete problem is not in* P.
2. *Every* NP*-complete problem is not in* P.
3. $\mathrm{P} \neq \mathrm{NP}$.

Satisfiability

A *propositional formula* is a sentence made from variables, operators $\wedge$, $\vee$, and $\neg$, and parentheses. An example is

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

A *3CNF formula* is a propositional formula consisting of the conjunction of a number of disjunctive clauses with at most 3 literals. An example of this is

$$\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_5) \wedge (\neg x_1 \vee x_4 \vee x_5)$$

Satisfiability

An *assignment* for a formula ϕ is a true/false setting of ϕ's variables.

An assignment *satisfies* ϕ if it makes ϕ evaluate to true in the standard way.

A formula ϕ is *satisfiable* if it has a satisfying assignment.

We define two satisfiability decision problems:

$$\mathrm{SAT} = \{\phi \,|\, \phi \text{ is a satisfiable formula}\}$$

$$\mathrm{3SAT} = \{\phi \,|\, \phi \text{ is a satisfiable 3CNF formula}\}$$

Cook-Levin Theorem (1971, 1973)

$\mathrm{3SAT}$ is NP-complete.

This requires showing that for every $A \in \mathrm{NP}$, $A \leq_{\mathrm{P}} \mathrm{3SAT}$.

Cook-Levin Theorem (1971, 1973)

$\mathrm{3SAT}$ is NP-complete.

This requires showing that for every $A \in \mathrm{NP}$, $A \leq_{\mathrm{P}} \mathrm{3SAT}$.

- There is a polynomial-time nondeterministic Turing machine for A.

Cook-Levin Theorem (1971, 1973)

$\mathrm{3SAT}$ is NP-complete.

This requires showing that for every $A \in \mathrm{NP}$, $A \leq_{\mathrm{P}} \mathrm{3SAT}$.

- There is a polynomial-time nondeterministic Turing machine for A.
- The reduction from A to $\mathrm{3SAT}$ expresses the computation of the nondeterministic Turing machine as a formula.

Cook-Levin Theorem (1971, 1973)

$\mathrm{3SAT}$ is NP-complete.

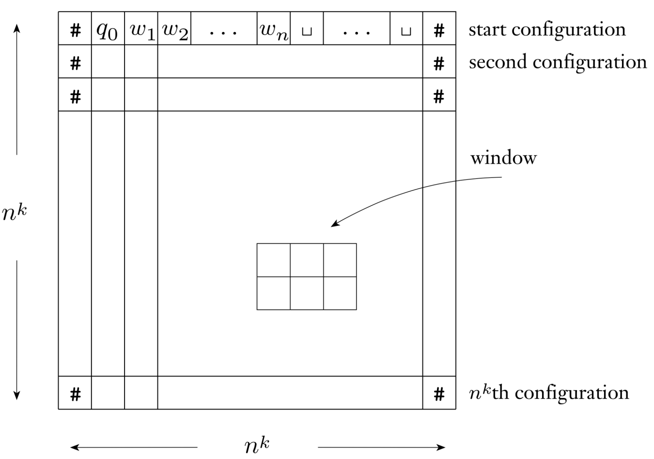
This requires showing that for every $A \in \mathrm{NP}$, $A \leq_{\mathrm{P}} \mathrm{3SAT}$.

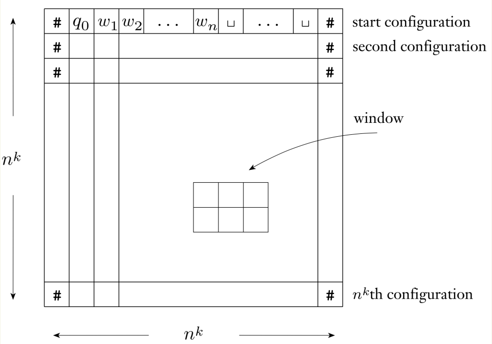
- There is a polynomial-time nondeterministic Turing machine for A.
- The reduction from A to $\mathrm{3SAT}$ expresses the computation of the nondeterministic Turing machine as a formula.
- We will first show $A \leq_{\mathrm{P}} \mathrm{SAT}$ and then we'll modify our proof to get $A \leq_{\mathrm{P}} \mathrm{3SAT}$.

Proof. Let $A \in \mathrm{NP}$. There is a nondeterministic TM that decides A in n^k time for some constant k.

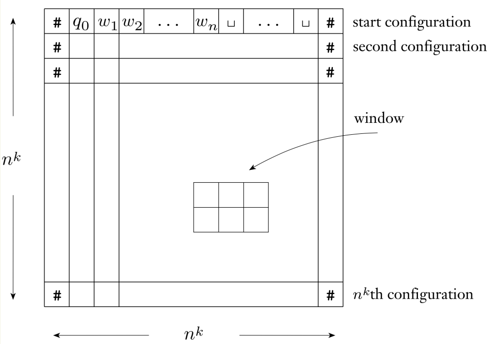
Proof. Let $A \in \mathrm{NP}$. There is a nondeterministic TM that decides A in n^k time for some constant k.

A *tableau* for N on w is an $n^k \times n^k$ table whose rows are the configurations of a branch of the computation of N on input w.

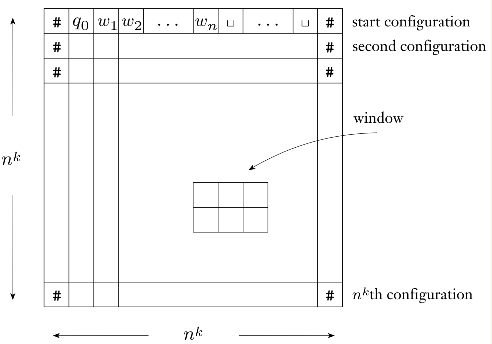




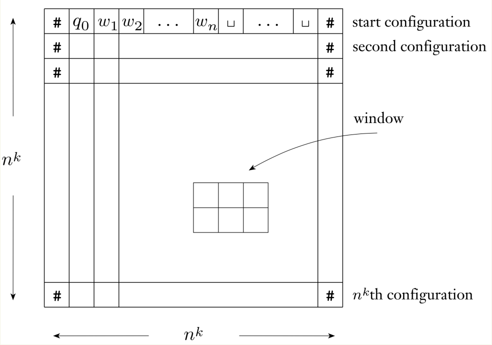
- For convenience later, we assume each configuration starts and ends with $\#$.



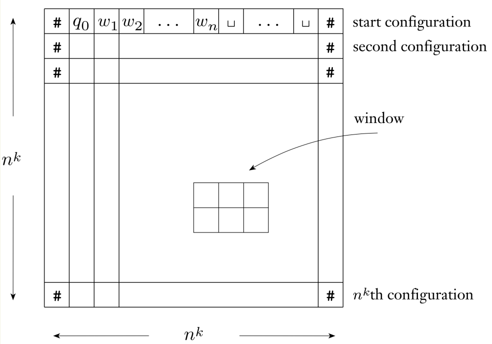
- For convenience later, we assume each configuration starts and ends with $\#$.
- The first row of the tableau is that start configuation of N on w.



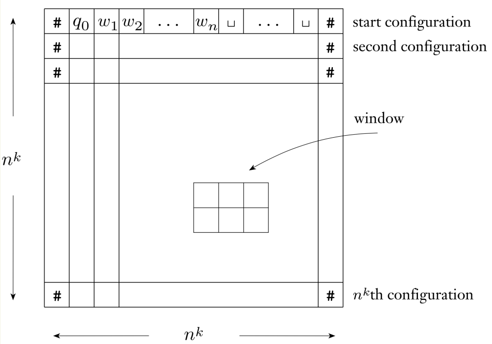
- For convenience later, we assume each configuration starts and ends with $\#$.
- The first row of the tableau is that start configuation of N on w.
- Each row follows the previous one according to N's transition function.



- For convenience later, we assume each configuration starts and ends with $\#$.
- The first row of the tableau is that start configuation of N on w.
- Each row follows the previous one according to N's transition function.
- A tableau is *accepting* if any row is an accepting configuration.

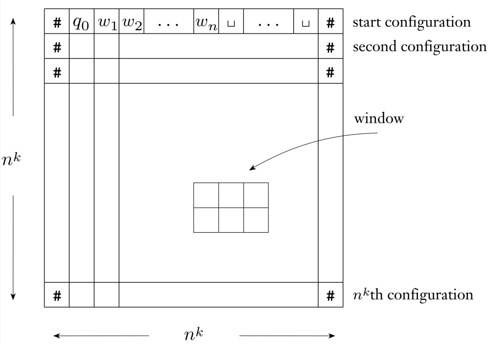


- For convenience later, we assume each configuration starts and ends with $\#$.
- The first row of the tableau is that start configuation of N on w.
- Each row follows the previous one according to N's transition function.
- A tableau is *accepting* if any row is an accepting configuration.
- Every accepting computation path of N corresponds to an accepting tableau.



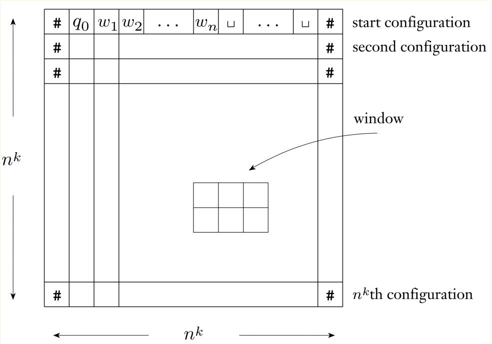
Our reduction f will take an input w and produce a formula ϕ.

- Let $C = Q \cup \Gamma \cup \{\#\}$, where Q is the set of states and Γ is the tape alphabet.



Our reduction f will take an input w and produce a formula ϕ.

- Let $C = Q \cup \Gamma \cup \{\#\}$, where Q is the set of states and Γ is the tape alphabet.
- For each $s \in C$ and $1 \leq i, j \leq n^k$ we have a variable $x_{i,j,s}$.

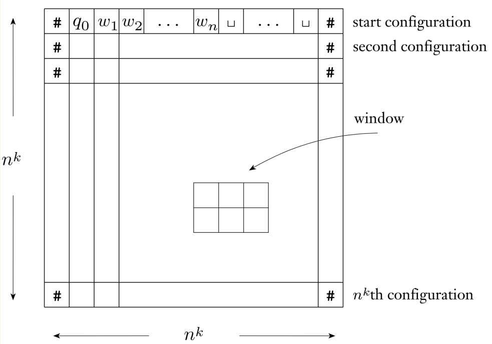


Our reduction f will take an input w and produce a formula ϕ.

- Let $C = Q \cup \Gamma \cup \{\#\}$, where Q is the set of states and Γ is the tape alphabet.

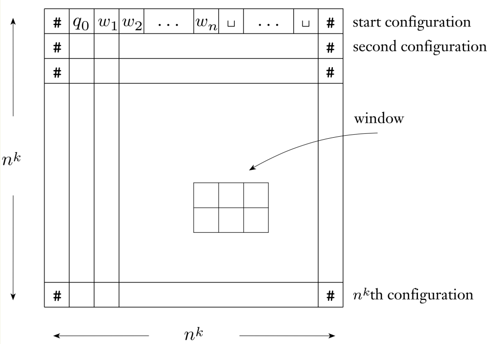
- For each $s \in C$ and $1 \leq i, j \leq n^k$ we have a variable $x_{i,j,s}$.

- Each of the $(n^k)^2$ entries of a tableau is called a *cell*.



Our reduction f will take an input w and produce a formula ϕ.

- Let $C = Q \cup \Gamma \cup \{\#\}$, where Q is the set of states and Γ is the tape alphabet.
- For each $s \in C$ and $1 \leq i, j \leq n^k$ we have a variable $x_{i,j,s}$.
- Each of the $(n^k)^2$ entries of a tableau is called a *cell*.
- *cell*$[i, j]$ is the cell in row i and column j.



Our reduction f will take an input w and produce a formula ϕ.

- Let $C = Q \cup \Gamma \cup \{\#\}$, where Q is the set of states and Γ is the tape alphabet.
- For each $s \in C$ and $1 \leq i, j \leq n^k$ we have a variable $x_{i,j,s}$.
- Each of the $(n^k)^2$ entries of a tableau is called a *cell*.
- $cell[i,j]$ is the cell in row i and column j.
- If $x_{i,j,s} = 1$, it means $cell[i,j]$ contains an s.

Our formula ϕ will be the conjunction of four parts:

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

We will define these four parts separately.

ϕ_{cell}

Recall the variable $x_{i,j,s}$ corresponds to placing an s in $cell[i,j]$.

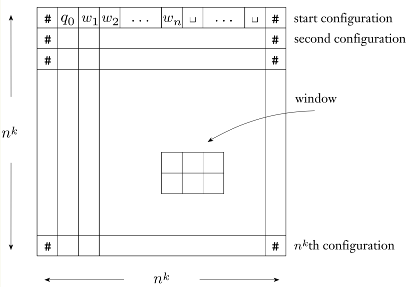
The formula ϕ_{cell} ensures that exactly one variable is turned on for each cell.

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

The first part ensures that some variable is turned on. The second part ensures that only one variable is turned on.

Note that ϕ_{cell} has size $O(n^{2k})$ and may be computed in time $O(n^{2k})$.

ϕ_{start}

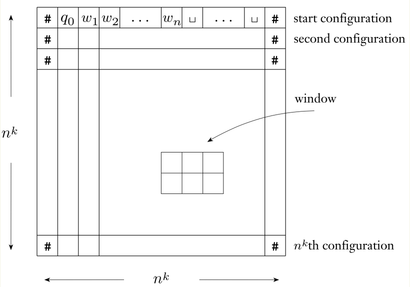


The formula ϕ_{start} ensures the first row of the tableau is the start configuration of N on w.

$$
\begin{aligned}
\phi_{\mathrm{start}} \quad = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\
& x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge \\
& x_{1,n+3,\sqcup} \wedge \cdots x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}
\end{aligned}
$$

Note that ϕ_{start} has size $O(n^k)$ and may be computed in time $O(n^k)$.

ϕ_{accept}

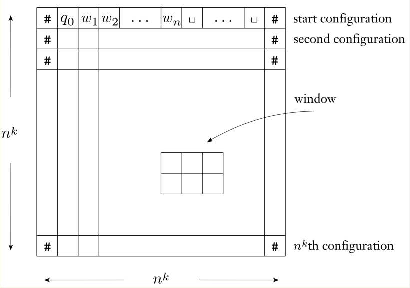


The formula ϕ_{accept} ensures that an accepting computation appears in the tableau. The accept state q_{accept} must appear in the tableau.

$$\phi_{\mathrm{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\mathrm{accept}}}$$

Note that ϕ_{accept} has size $O(n^{2k})$ and may be computed in time $O(n^{2k})$.

ϕ_{move}



The formula ϕ_{move} ensures that each row of tableau is a configuration that follows the configuration in the previous row by one move of N's transition function. The formula ensures that each 2×3 window of cells is legal.

ϕ_{move}

Suppose $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$.

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

FIGURE 7.39
Examples of legal windows

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

FIGURE 7.40
Examples of illegal windows

ϕ_{move}

The (i,j)-window has $cell[i,j]$ in the upper central position. Let the six cells of the window be $a_1,\ldots,a_6$. Then we define

$$\psi_{i,j} = \bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(\begin{array}{l} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge \\ x_{i+1,j-1,a_1} \wedge x_{i+1,j,a_2} \wedge x_{i+1,j+1,a_3} \end{array} \right)$$

ϕ_{move}

The (i, j)-window has *cell*$[i, j]$ in the upper central position. Let the six cells of the window be $a_1, \ldots, a_6$. Then we define

$$\psi_{i,j} = \bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \begin{pmatrix} x_{i,j-1,a_1} \ \wedge \ x_{i,j,a_2} \wedge x_{i,j+1,a_3} \ \wedge \\ x_{i+1,j-1,a_1} \ \wedge \ x_{i+1,j,a_2} \ \wedge \ x_{i+1,j+1,a_3} \end{pmatrix}$$

Finally, we define

$$\phi_{\mathrm{move}} = \bigwedge_{\substack{1 \le i \le n^k \\ 1 < j < n^k}} \psi_{i,j}.$$

Note that ϕ_{move} has size $O(n^{2k})$ and may be computed in time $O(n^{2k})$.

Our reduction f outputs the formula

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}.$$

Each of the four parts of ϕ is computable in $O(n^{2k})$ time, so ϕ is computable in $O(n^{2k})$ time. Thus f is polynomial-time computable.

We have

$$\begin{aligned} w \in A \quad &\iff \quad N \text{ accepts } w \\ &\iff \quad \text{there is an accepting computation history of } N \text{ on } w \\ &\iff \quad \text{there is an accepting tableau for } N \text{ on } w \\ &\iff \quad \phi \text{ is satisfiable.} \end{aligned}$$

Therefore $A \leq_{\mathrm{m}} \mathrm{SAT}$ via f.

We now modify the reduction to get $A \leq_{\mathrm{m}} 3\mathrm{SAT}$. In

$$\phi = \phi_{\mathrm{cell}} \wedge \phi_{\mathrm{start}} \wedge \phi_{\mathrm{move}} \wedge \phi_{\mathrm{accept}},$$

the formulas ϕ_{cell}, ϕ_{start}, and ϕ_{accept} are in CNF. We can use the distributive law to put ϕ_{move} into CNF form as well.

We now modify the reduction to get $A \leq_{\mathrm{m}} 3\mathrm{SAT}$. In

$$\phi = \phi_{\mathrm{cell}} \wedge \phi_{\mathrm{start}} \wedge \phi_{\mathrm{move}} \wedge \phi_{\mathrm{accept}},$$

the formulas ϕ_{cell}, ϕ_{start}, and ϕ_{accept} are in CNF. We can use the distributive law to put ϕ_{move} into CNF form as well.

To get 3CNF formulas we use additional variables to split each clause with more than 3 literals into multiple clauses that have 3 literals. For example, the clause $(a_1 \vee a_2 \vee a_3 \vee a_4)$ is replaced by $(a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$, where z is a new variable.

We now modify the reduction to get $A \leq_{\mathrm{m}} \mathrm{3SAT}$. In

$$\phi = \phi_{\mathrm{cell}} \wedge \phi_{\mathrm{start}} \wedge \phi_{\mathrm{move}} \wedge \phi_{\mathrm{accept}},$$

the formulas ϕ_{cell}, ϕ_{start}, and ϕ_{accept} are in CNF. We can use the distributive law to put ϕ_{move} into CNF form as well.

To get 3CNF formulas we use additional variables to split each clause with more than 3 literals into multiple clauses that have 3 literals. For example, the clause $(a_1 \vee a_2 \vee a_3 \vee a_4)$ is replaced by $(a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$, where z is a new variable.

More generally, a clause

$$(a_1 \vee a_2 \vee \ldots \vee a_l)$$

with l literals is replaced by $l - 2$ clauses:

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_2 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l)$$

where the z_i's are new variables. $\Box$

We have shown that 3SAT is NP-complete. We can use the following lemma to build on this to show more problems are NP-complete.

Lemma

$\leq_{\mathrm{P}}$ is transitive. That is, if $A \leq_{\mathrm{P}} B$ and $B \leq_{\mathrm{P}} C$, then $A \leq_{\mathrm{P}} C$.

We have shown that 3SAT is NP-complete. We can use the following lemma to build on this to show more problems are NP-complete.

Lemma

$\leq_{\mathrm{P}}$ is transitive. That is, if $A \leq_{\mathrm{P}} B$ and $B \leq_{\mathrm{P}} C$, then $A \leq_{\mathrm{P}} C$.

Proof. Let f be the reduction from A to B and let g be the reduction from B to C. Then

- $I \in A \Leftrightarrow f(I) \in B$ for all instances I,
- $J \in B \Leftrightarrow g(J) \in C$ for all instances J,
- f is computable in some polynomial $p(n)$ time, and
- g is computable in some polynomial $q(n)$ time.

Define

$$h(I) = g(f(I)).$$

Define

$$h(I) = g(f(I)).$$

Then for every instance I,

$$I \in A \quad \Leftrightarrow \quad f(I) \in B$$

Define

$$h(I) = g(f(I)).$$

Then for every instance I,

$$\begin{aligned} I \in A \quad &\Leftrightarrow \quad f(I) \in B \\ &\Leftrightarrow \quad g(f(I)) \in C \end{aligned}$$

Define

$$h(I) = g(f(I)).$$

Then for every instance I,

$$\begin{aligned} I \in A &\Leftrightarrow f(I) \in B \\ &\Leftrightarrow g(f(I)) \in C \\ &\Leftrightarrow h(I) \in C. \end{aligned}$$

Define

$$h(I) = g(f(I)).$$

Then for every instance I,

$$\begin{aligned} I \in A &\Leftrightarrow f(I) \in B \\ &\Leftrightarrow g(f(I)) \in C \\ &\Leftrightarrow h(I) \in C. \end{aligned}$$

Let $n = |I|$. To compute $h(I)$ we need at most

- $p(n)$ time to compute $f(I)$.
- $q(p(n))$ time to compute $g(f(I))$ because $|f(I)| \le p(n)$.

The total time $p(n) + q(p(n))$ is polynomial.

Define
$$h(I) = g(f(I)).$$
Then for every instance I,
$$\begin{aligned} I \in A &\Leftrightarrow f(I) \in B \\ &\Leftrightarrow g(f(I)) \in C \\ &\Leftrightarrow h(I) \in C. \end{aligned}$$
Let $n = |I|$. To compute $h(I)$ we need at most

- $p(n)$ time to compute $f(I)$.
- $q(p(n))$ time to compute $g(f(I))$ because $|f(I)| \leq p(n)$.

The total time $p(n) + q(p(n))$ is polynomial.

Therefore $A \leq_{\mathrm{P}} C$ via h. $\square$

Corollary

Let $A, B \in \mathrm{NP}$*. If* A *is* NP*-complete and* $A \leq_{\mathrm{P}} B$*, then* B *is also* NP*-complete.*

Corollary

Let $A, B \in \mathrm{NP}$. If A is NP-complete and $A \leq_{\mathrm{P}} B$, then B is also NP-complete.

Proof. Because A is NP-complete, $C \leq_{\mathrm{P}} A$ for every $C \in \mathrm{NP}$. By transitivity and $A \leq_{\mathrm{P}} B$, we have $C \leq_{\mathrm{P}} B$ for every $B \in \mathrm{NP}$. $\square$

Corollary

Let $A, B \in \mathrm{NP}$. If A is NP-complete and $A \leq_{\mathrm{P}} B$, then B is also NP-complete.

Proof. Because A is NP-complete, $C \leq_{\mathrm{P}} A$ for every $C \in \mathrm{NP}$. By transitivity and $A \leq_{\mathrm{P}} B$, we have $C \leq_{\mathrm{P}} B$ for every $B \in \mathrm{NP}$. $\square$

To prove that a problem $B \in \mathrm{NP}$ is NP-complete, we don't have to reduce every problem in NP to it. We may select any NP-complete problem A and show that $A \leq_{\mathrm{P}} B$.

NP-Completeness Proofs

Here is the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.

NP-Completeness Proofs

Here is the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.
2. Choose some NP-complete problem A that is a candidate for reduction to B. Often we try to find a problem that is similar to B.

NP-Completeness Proofs

Here is the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.
2. Choose some NP-complete problem A that is a candidate for reduction to B. Often we try to find a problem that is similar to B.
3. Reduce A to B.

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
   $$I \mapsto I'$$

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
   $$I \mapsto I'$$
   Then we need to prove that for every instance I of A,
   $$I \in A \iff I' \in B.$$

3. Reduce A to B.

   For this, we need to define a mapping of instances I of A to instances I' of B.

   $$I \mapsto I'$$

   Then we need to prove that for every instance I of A,

   $$I \in A \iff I' \in B.$$

   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.

3. Reduce A to B.

   For this, we need to define a mapping of instances I of A to instances I' of B.

   $$I \mapsto I'$$

   Then we need to prove that for every instance I of A,

   $$I \in A \iff I' \in B.$$

   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.

   - To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.

3. Reduce A to B.

   For this, we need to define a mapping of instances I of A to instances I' of B.

   $$I \mapsto I'$$

   Then we need to prove that for every instance I of A,

   $$I \in A \iff I' \in B.$$

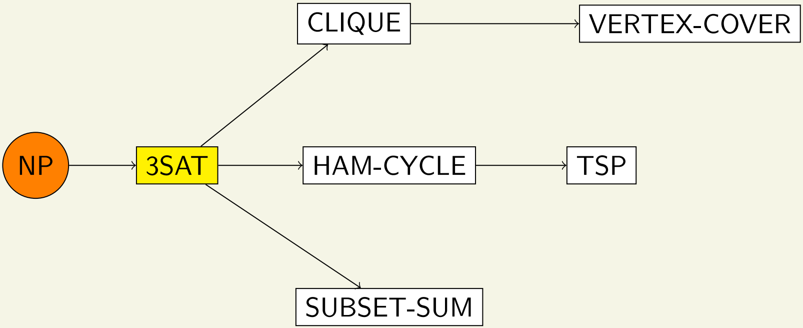
   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.

   - To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.
   - To show the converse, assume that $I' \in B$. Then there is some witness w' for I'. We use w' to construct a witness w that shows $I \in A$.

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
$$I \mapsto I'$$
   Then we need to prove that for every instance I of A,
$$I \in A \iff I' \in B.$$
   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.
   - To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.
   - To show the converse, assume that $I' \in B$. Then there is some witness w' for I'. We use w' to construct a witness w that shows $I \in A$.

   Finally, we have to verify that the mapping is polynomial-time computable. Usually this doesn't require formal analysis of the algorithm.

Plan for NP-completeness reductions:



Clique

In a graph $G = (V, E)$, a *clique* is a fully interconnected subset of the vertices. That is, a clique is a set $V' \subseteq V$ such that for all $u, v \in V'$, $(u, v) \in E$. The clique decision problem is

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

In other words, given a graph G and number k, does G have a clique of size k?

Clique

In a graph $G = (V, E)$, a *clique* is a fully interconnected subset of the vertices. That is, a clique is a set $V' \subseteq V$ such that for all $u, v \in V'$, $(u, v) \in E$. The clique decision problem is

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

In other words, given a graph G and number k, does G have a clique of size k?

Notice that we can solve this with a brute force search through all subgraphs of size k, which looks like polynomial time $O(n^k)$, but when $k = \Omega(n)$, say $n/3$, the time required is exponential.

Clique

In a graph $G = (V, E)$, a *clique* is a fully interconnected subset of the vertices. That is, a clique is a set $V' \subseteq V$ such that for all $u, v \in V'$, $(u, v) \in E$. The clique decision problem is

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

In other words, given a graph G and number k, does G have a clique of size k?

Notice that we can solve this with a brute force search through all subgraphs of size k, which looks like polynomial time $O(n^k)$, but when $k = \Omega(n)$, say $n/3$, the time required is exponential.

However, it is easy to see that $\mathrm{CLIQUE} \in \mathrm{NP}$. The clique itself is a witness. So $\langle G, k\rangle \in \mathrm{CLIQUE} \iff (\exists V')|V'| = k$ and V' is a clique in G.

Theorem

CLIQUE *is* NP*-complete.*

Proof. We will show $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{CLIQUE}$. Let $\phi = C_1 \wedge \ldots \wedge C_k$ be a 3CNF formula, where each C_i is a clause with at most 3 literals. We construct a graph G as follows.

Theorem

CLIQUE *is* NP*-complete.*

Proof. We will show $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{CLIQUE}$. Let $\phi = C_1 \wedge \ldots \wedge C_k$ be a 3CNF formula, where each C_i is a clause with at most 3 literals. We construct a graph G as follows.

For each clause $C_i = \left(l_1^{(i)} \vee l_2^{(i)} \vee l_3^{(i)}\right)$ we put vertices $v_1^{(i)}$, $v_2^{(i)}$, $v_3^{(i)}$ in G. (If C_i has 1 or 2 literals, we put 1 or 2 vertices in G.)

Theorem

CLIQUE *is* NP*-complete.*

Proof. We will show $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{CLIQUE}$. Let $\phi = C_1 \wedge \ldots \wedge C_k$ be a 3CNF formula, where each C_i is a clause with at most 3 literals. We construct a graph G as follows.

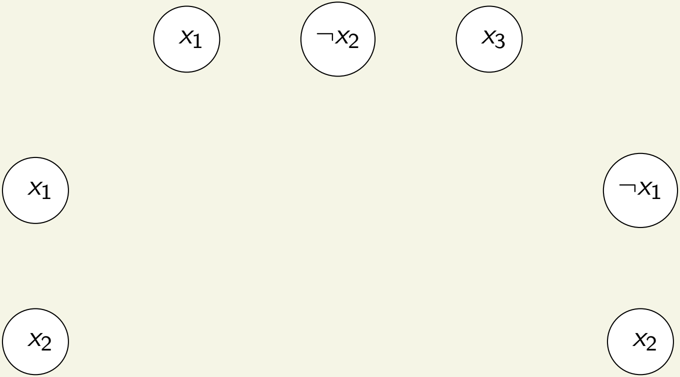
For each clause $C_i = \left(l_1^{(i)} \vee l_2^{(i)} \vee l_3^{(i)}\right)$ we put vertices $v_1^{(i)}$, $v_2^{(i)}$, $v_3^{(i)}$ in G. (If C_i has 1 or 2 literals, we put 1 or 2 vertices in G.) We put an edge between $v_a^{(i)}$ and $v_b^{(j)}$ if the following two conditions both hold.

1. $i \neq j$. In other words, $v_a^{(i)}$ and $v_b^{(j)}$ correspond to literals in different clauses.
2. $l_a^{(i)} \neq \neg l_b^{(j)}$. In other words, the corresponding literals are consistent.

For example, consider the formula

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

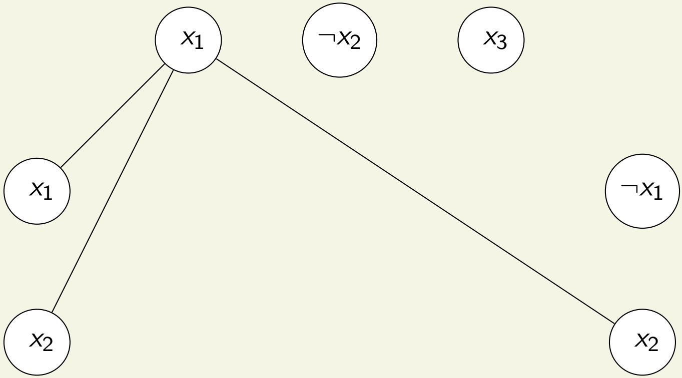
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



For example, consider the formula

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

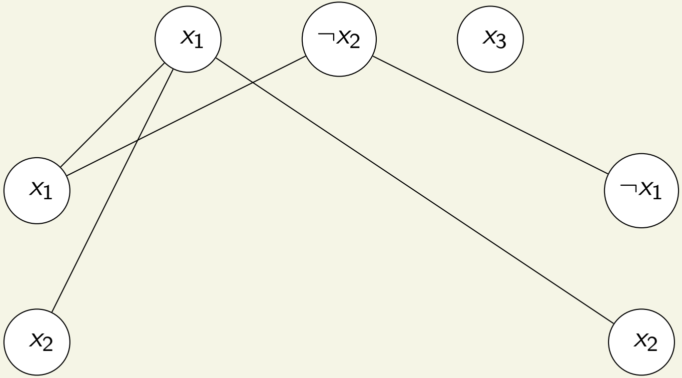
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



For example, consider the formula

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

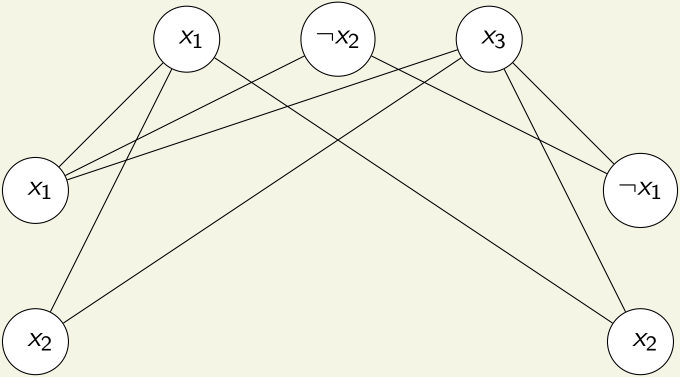
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



For example, consider the formula

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

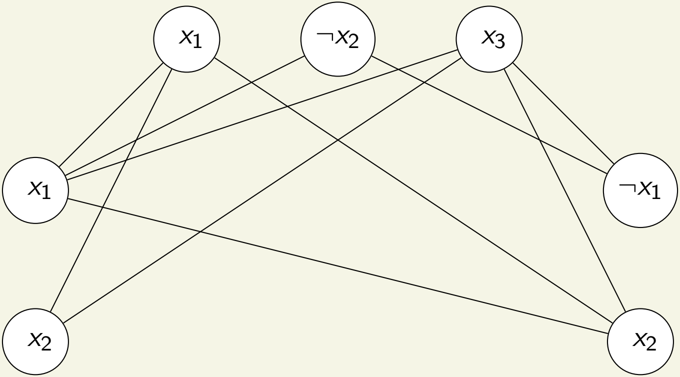
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



For example, consider the formula

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

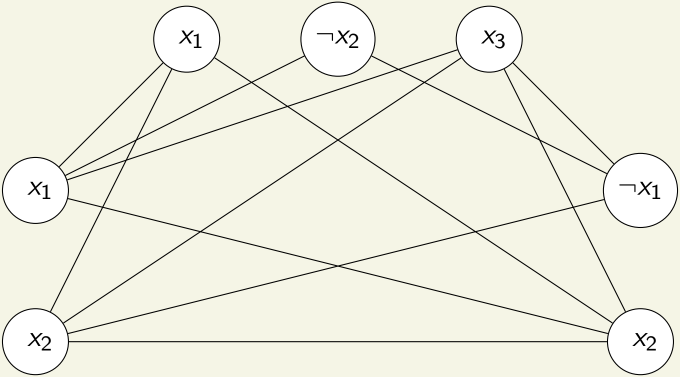
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



For example, consider the formula

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

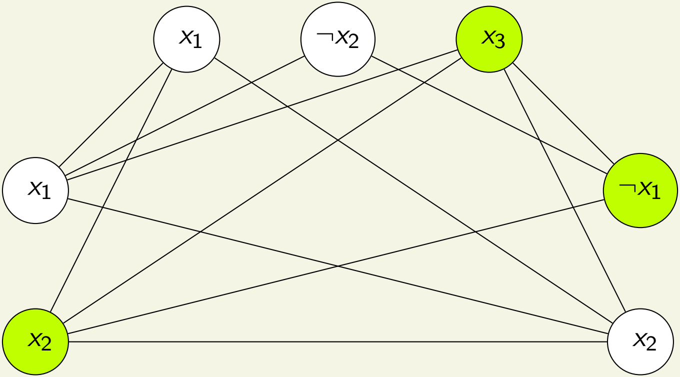
The graph G appears below. The three vertices at the top correspond to the first clause, the two at the left are for the second clause, and the two at the right are for the third clause.



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

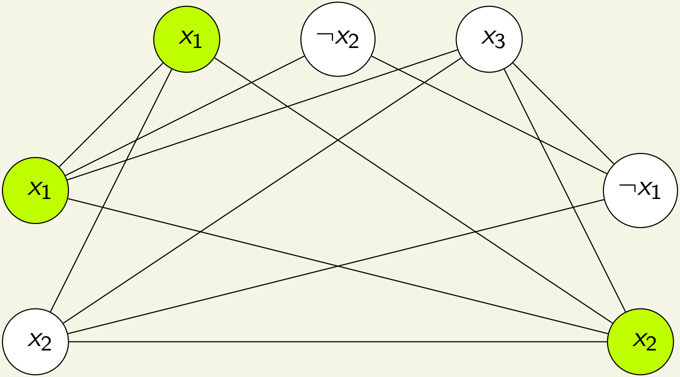
Given a satisfying assignment: $x_1 = F, x_2 = T, x_3 = T$

There is a corresponding clique:



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

Given a clique:



There is a corresponding satisfying assignment:
$x_1 = T, x_2 = T, x_3 = T$ (value of x_3 doesn't matter)

Suppose that ϕ is satisfiable. Then there is an assignment to the variables that makes at least one literal true in each clause.

Suppose that ϕ is satisfiable. Then there is an assignment to the variables that makes at least one literal true in each clause.

Choose one such literal from each clause and let V' be the corresponding set of k vertices.

Suppose that ϕ is satisfiable. Then there is an assignment to the variables that makes at least one literal true in each clause.

Choose one such literal from each clause and let V' be the corresponding set of k vertices.

Then V' is a clique of size k because our construction ensures there is an edge between each vertex in V'.

Suppose that ϕ is satisfiable. Then there is an assignment to the variables that makes at least one literal true in each clause.

Choose one such literal from each clause and let V' be the corresponding set of k vertices.

Then V' is a clique of size k because our construction ensures there is an edge between each vertex in V'.

This shows $\phi \in \mathrm{3SAT} \Rightarrow \langle G, k \rangle \in \mathrm{CLIQUE}$.

Conversely, suppose that G has a clique of size k. Let V' be this clique.

Conversely, suppose that G has a clique of size k. Let V' be this clique.

Then V' must contain a vertex from each of the k groups that correspond to the clauses. This is because there cannot be edges between vertices in the same group.

Conversely, suppose that G has a clique of size k. Let V' be this clique.

Then V' must contain a vertex from each of the k groups that correspond to the clauses. This is because there cannot be edges between vertices in the same group.

From V' we obtain a satisfying assignment for ϕ by setting the variables to make each of the corresponding literals true.

- We can do this because there is no edge between a literal and its negation, thus we won't ever try to set a variable to both true and false.
- If a variable does not correspond to a vertex in V', then we can set it arbitrarily.

Conversely, suppose that G has a clique of size k. Let V' be this clique.

Then V' must contain a vertex from each of the k groups that correspond to the clauses. This is because there cannot be edges between vertices in the same group.

From V' we obtain a satisfying assignment for ϕ by setting the variables to make each of the corresponding literals true.

- We can do this because there is no edge between a literal and its negation, thus we won't ever try to set a variable to both true and false.
- If a variable does not correspond to a vertex in V', then we can set it arbitrarily.

This assignment satisfies at least one literal in each clause of ϕ. Therefore, $\langle G, k\rangle \in \mathrm{CLIQUE} \Rightarrow \phi \in \mathrm{3SAT}$.

Conversely, suppose that G has a clique of size k. Let V' be this clique.

Then V' must contain a vertex from each of the k groups that correspond to the clauses. This is because there cannot be edges between vertices in the same group.

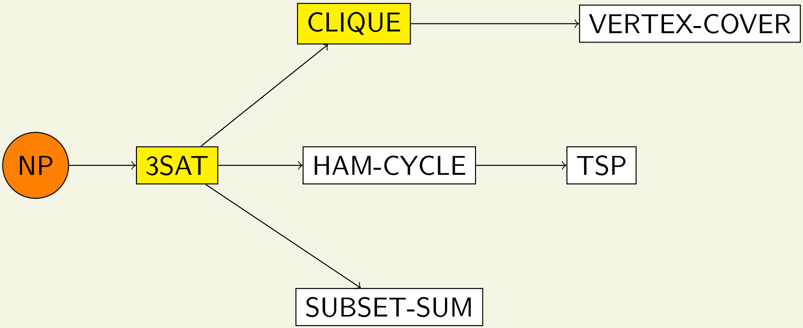
From V' we obtain a satisfying assignment for ϕ by setting the variables to make each of the corresponding literals true.

- We can do this because there is no edge between a literal and its negation, thus we won't ever try to set a variable to both true and false.
- If a variable does not correspond to a vertex in V', then we can set it arbitrarily.

This assignment satisfies at least one literal in each clause of ϕ. Therefore, $\langle G, k \rangle \in \mathrm{CLIQUE} \Rightarrow \phi \in \mathrm{3SAT}$.

Since the mapping $\phi \mapsto \langle G, k \rangle$ can be computed in polynomial time, we have that $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{CLIQUE}$. $\square$

Plan for NP-completeness reductions:



Vertex Cover

A *vertex cover* of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). In other words, a vertex cover is a subset of the vertices that touches all edges in the graph. The decision problem is

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

Vertex Cover

A *vertex cover* of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). In other words, a vertex cover is a subset of the vertices that touches all edges in the graph. The decision problem is

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

Note that V, the set of all vertices in the graph, is always a vertex cover. Also, if V' is a vertex cover and $V'' \supseteq V'$, then V'' is also a vertex cover. The goal is to determine if small vertex covers exist.

Theorem

$\mathrm{VERTEX\text{-}COVER}$ *is* NP*-complete.*

Proof. To see that $\mathrm{VERTEX\text{-}COVER} \in \mathrm{NP}$, note that $\langle G, k\rangle \in \mathrm{VERTEX\text{-}COVER} \iff (\exists V')|V'| \leq k$ and V' is a vertex cover for G. The set V' is the witness and all we have to do is check if V' covers all the edges, which can be done in polynomial time.

Theorem

$\mathrm{VERTEX\text{-}COVER}$ *is* NP*-complete.*

Proof. To see that $\mathrm{VERTEX\text{-}COVER} \in \mathrm{NP}$, note that $\langle G, k\rangle \in \mathrm{VERTEX\text{-}COVER} \iff (\exists V')|V'| \leq k$ and V' is a vertex cover for G. The set V' is the witness and all we have to do is check if V' covers all the edges, which can be done in polynomial time.

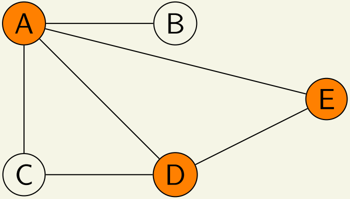
To show completeness, we will reduce CLIQUE to $\mathrm{VERTEX\text{-}COVER}$. Given a graph $G = (V, E)$, we define the complementary graph $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{(u, v) \in V \times V \,|(u, v) \notin E \text{ and } u \neq v\}\,.$$

We claim that G has a clique of size k if and only if $\bar{G}$ has a vertex cover of size $|V| - k$.

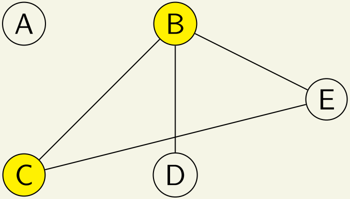
Examples

G:



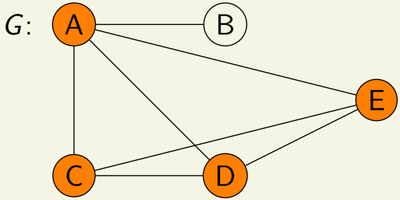
$\{A, D, E\}$ is a clique of size 3

$\overline{G}$:

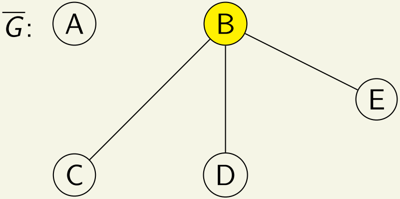


$\{B, C\}$ is a vertex cover of size 2

Examples



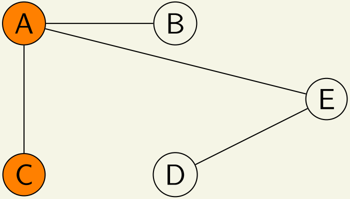
$\{A, C, D, E\}$ is a clique of size 4



$\{B\}$ is a vertex cover of size 1

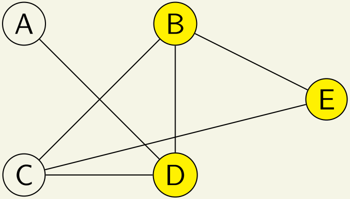
Examples

G:



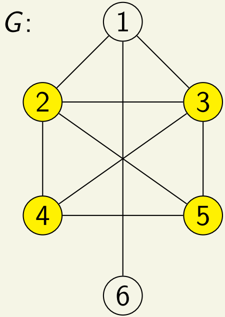
$\{A, C\}$ is a clique of size 2

$\overline{G}$:

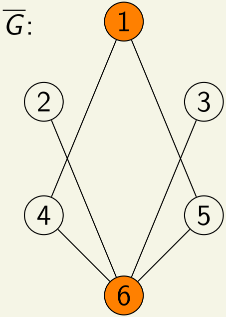


$\{B, D, E\}$ is a vertex cover of size 3

Examples



$V' = \{2, 3, 4, 5\}$
is a clique of size 4



$V - V' = \{1, 6\}$
is a vertex cover of size 2

Suppose that G has a clique V' with $|V'| = k$. We claim that $V - V'$ is a vertex cover in $\bar{G}$.

Suppose that G has a clique V' with $|V'| = k$. We claim that $V - V'$ is a vertex cover in $\bar{G}$.

To see this, let $(u, v) \in \bar{E}$ be any edge in $\bar{G}$. Then $(u, v) \notin E$, so at least one of the two vertices is not in the clique: at least one of u, v is not in V'. Therefore at least one of u or v is in $V - V'$.

Suppose that G has a clique V' with $|V'| = k$. We claim that $V - V'$ is a vertex cover in $\bar{G}$.

To see this, let $(u, v) \in \bar{E}$ be any edge in $\bar{G}$. Then $(u, v) \notin E$, so at least one of the two vertices is not in the clique: at least one of u, v is not in V'. Therefore at least one of u or v is in $V - V'$.

Since $(u, v) \in \bar{E}$ was arbitrary, we have shown that $V - V'$ is a vertex cover in $\bar{G}$.

Suppose that G has a clique V' with $|V'| = k$. We claim that $V - V'$ is a vertex cover in $\bar{G}$.

To see this, let $(u, v) \in \bar{E}$ be any edge in $\bar{G}$. Then $(u, v) \notin E$, so at least one of the two vertices is not in the clique: at least one of u, v is not in V'. Therefore at least one of u or v is in $V - V'$.

Since $(u, v) \in \bar{E}$ was arbitrary, we have shown that $V - V'$ is a vertex cover in $\bar{G}$.

Since $|V - V'| = |V| - k$, this shows

$$\langle G, k\rangle \in \mathrm{CLIQUE} \Rightarrow \langle \bar{G}, |V| - k\rangle \in \text{VERTEX-COVER}.$$

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

If we take the contrapositive of the implication, we have that for all $u, v \in V$,

$$u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \bar{E}.$$

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

If we take the contrapositive of the implication, we have that for all $u, v \in V$,

$$u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \bar{E}.$$

This is equivalent to saying that for all u, v with $u \neq v$,

$$u, v \in V - V' \Rightarrow (u, v) \in E,$$

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

If we take the contrapositive of the implication, we have that for all $u, v \in V$,

$$u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \bar{E}.$$

This is equivalent to saying that for all u, v with $u \neq v$,

$$u, v \in V - V' \Rightarrow (u, v) \in E,$$

which means that $V - V'$ is a clique in G.

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

If we take the contrapositive of the implication, we have that for all $u, v \in V$,

$$u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \bar{E}.$$

This is equivalent to saying that for all u, v with $u \neq v$,

$$u, v \in V - V' \Rightarrow (u, v) \in E,$$

which means that $V - V'$ is a clique in G. Notice that

$$|V - V'| = |V| - |V'| = |V| - |V| + k = k,$$

so G has a clique of size k.

Now suppose that $\bar{G}$ has a vertex cover V' with $|V'| = |V| - k$. Then for all $u, v \in V$, we have

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'.$$

If we take the contrapositive of the implication, we have that for all $u, v \in V$,

$$u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \bar{E}.$$

This is equivalent to saying that for all u, v with $u \neq v$,

$$u, v \in V - V' \Rightarrow (u, v) \in E,$$

which means that $V - V'$ is a clique in G. Notice that

$$|V - V'| = |V| - |V'| = |V| - |V| + k = k,$$

so G has a clique of size k. This shows

$$\langle \bar{G}, |V| - k \rangle \in \text{VERTEX-COVER} \Rightarrow \langle G, k \rangle \in \text{CLIQUE}.$$

We have shown that for any graph $G = (V, E)$ and number k,

$$\langle G, k \rangle \in \mathrm{CLIQUE} \iff \langle \bar{G}, |V| - k \rangle \in \mathrm{VERTEX\text{-}COVER}.$$

Since the mapping $\langle G, k \rangle \mapsto \langle \bar{G}, |V| - k \rangle$ can be computed in polynomial time, we have $\mathrm{CLIQUE} \leq_{\mathrm{P}} \mathrm{VERTEX\text{-}COVER}$. $\square$

Plan for NP-completeness reductions: