Algorithms and Computability

Lecture 13: NP-Complete Problems

Christian Schilling (christianms@cs.aau.dk)

slides courtesy of Martin Zimmermann

Last Time in Algorithms and Computability

We have seen:

- The complexity class NP: The class of languages accepted by polynomial-time NTMs
- Polynomial-time reductions

Recall: $\mathrm{P} \subseteq \mathrm{NP}$

The (literally) Million Dollar Question

Is verifying certificates easier than finding certificates or not:

$\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \subsetneq \mathrm{NP}$?

- One of the most challenging and most important questions of (theoretical) computer science

Polynomial-Time Reductions

Definition

Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages. We say that A is polynomial-time reducible to B, written $A \leq_P B$, if and only if

1. there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that
2. for every $w \in \Sigma_1^*$: $w \in A \Leftrightarrow f(w) \in B$

Theorem

Let $A \leq_P B$

1. *If* $B \in \mathrm{P}$, *then* $A \in \mathrm{P}$
2. *If* $B \in \mathrm{NP}$, *then* $A \in \mathrm{NP}$

Polynomial-Time Reductions

Definition

Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages. We say that A is polynomial-time reducible to B, written $A \leq_P B$, if and only if

1. there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that
2. for every $w \in \Sigma_1^*$: $w \in A \Leftrightarrow f(w) \in B$

Theorem

Let $A \leq_P B$

1. *If* $B \in \mathrm{P}$, *then* $A \in \mathrm{P}$
2. *If* $B \in \mathrm{NP}$, *then* $A \in \mathrm{NP}$

This result can be used to prove **upper** bounds

What About Lower Bounds?

Recall:

Theorem

Let $A \leq_m B$

- *If* A *is not computable, then* B *is also not computable*
- *If* A *is not computably-enumerable, then* B *is also not computably-enumerable*

What About Lower Bounds?

Recall:

Theorem

Let $A \leq_m B$

- *If A is not computable, then B is also not computable*
- *If A is not computably-enumerable, then B is also not computably-enumerable*

- Today, we will study how to obtain analogous results for P and NP
- However, the fact that we have no problem that is proven to be in $\mathrm{NP} \setminus \mathrm{P}$ means we will identify candidates for such problems: the hardest problems in NP
- **Caveat:** These hardest problem might still be in P – we simply do not know
- We will discuss this in more detail later today

Agenda

We consider two types of graphs:

1. Directed Graphs: $G = (V, E)$ where
   - V is a finite set of vertices and
   - $E \subseteq V \times V$ is a set of (directed) edges

Example

(V, E) with $V = \{1, 2, 3, 4\}$ and
$E = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$:

$$\begin{tikzcd} 4 \arrow[d] & 3 \arrow[l] \\ 1 \arrow[r] & 2 \arrow[u] \end{tikzcd}$$

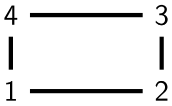
Reminder: Graphs

We consider two types of graphs:

1. Directed Graphs: $G = (V, E)$ where
   - V is a finite set of vertices and
   - $E \subseteq V \times V$ is a set of (directed) edges
2. Undirected Graphs: Directed graphs (V, E) such that $(v, v') \in E$ implies $(v', v) \in E$

Example

(V, E) with $V = \{1, 2, 3, 4\}$ and
$E = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (4, 1), (1, 4)\}$:



Note that we drop the arrow tips in undirected graphs

Reminder: The Clique Problem

You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

Reminder: The Clique Problem

You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

Reminder: The Clique Problem

You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

Reminder: The Clique Problem

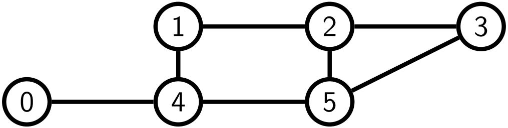
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



Reminder: The Clique Problem

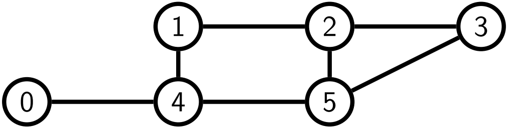
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A *k-clique* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$

Reminder: The Clique Problem

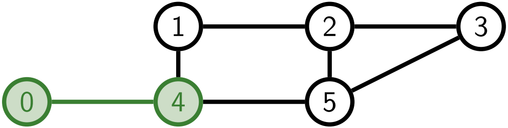
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A *k-clique* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$ ✓

Reminder: The Clique Problem

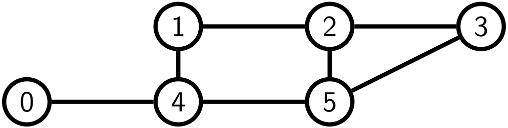
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$ ✓
$(G, 3) \in \mathrm{CLIQUE}?$

Reminder: The Clique Problem

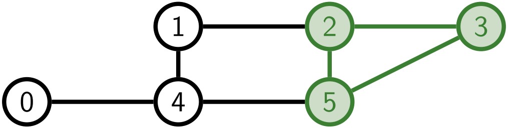
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A *k-clique* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$ ✓
$(G, 3) \in \mathrm{CLIQUE}?$ ✓

Reminder: The Clique Problem

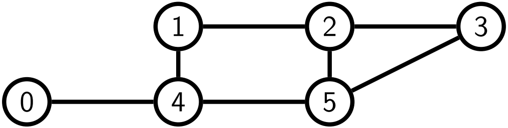
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$ ✓
$(G, 3) \in \mathrm{CLIQUE}?$ ✓
$(G, 4) \in \mathrm{CLIQUE}?$

Reminder: The Clique Problem

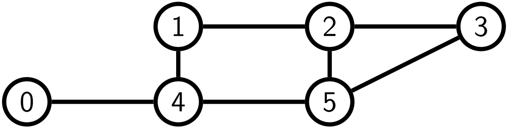
You work in the IT department of "Turing's Pizza Place" which prides itself with its 100 toppings. Your boss gives you a list of pairs of toppings that may go together and asks you to determine whether they can offer a pizza with 10 different toppings

- Let $G = (V, E)$ be an undirected graph. A k-clique is a subset $C \subseteq V$ of $k = |C|$ vertices such that $(v, v') \in E$ for all $v, v' \in C$ (i.e., a complete subgraph of size k)
- Decision problem:

$\mathrm{CLIQUE} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

- $\mathrm{CLIQUE} \in \mathrm{NP}$ (shown previously)

G :



$(G, 2) \in \mathrm{CLIQUE}?$ ✓
$(G, 3) \in \mathrm{CLIQUE}?$ ✓
$(G, 4) \in \mathrm{CLIQUE}?$ ✗

Vertex Cover

You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

Vertex Cover

You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A k-vertex cover is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

Vertex Cover

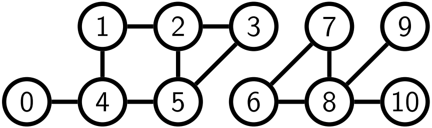
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



Vertex Cover

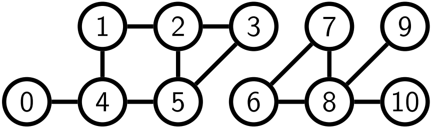
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}$?

Vertex Cover

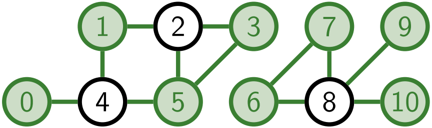
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}$? ✓

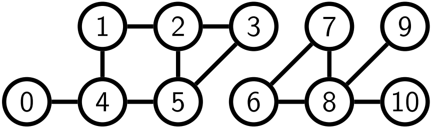
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}?$ ✓

$(G, 5) \in \mathrm{VC}?$

Vertex Cover

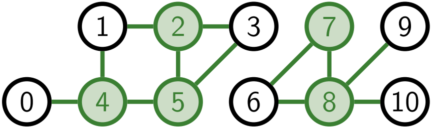
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}$? ✓

$(G, 5) \in \mathrm{VC}$? ✓

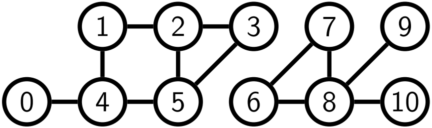
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}$? ✓
$(G, 5) \in \mathrm{VC}$? ✓
$(G, 4) \in \mathrm{VC}$?

Vertex Cover

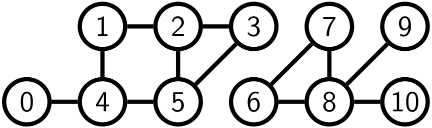
You are still working at that pizza place and have to plan the Christmas party: Each of the 200 employees has two favorite types of pizza, and the boss asks you whether you can pick 10 types so that everybody has at least one of their favorites among them

- Let $G = (V, E)$ be an undirected graph. A *k-vertex cover* is a subset $C \subseteq V$ of $k = |C|$ vertices such that $v \in C$ or $v' \in C$ for all $(v, v') \in E$
- Decision problem:

$$\mathrm{VC} = \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-vertex cover}\}$$

- Exercise: $\mathrm{VC} \in \mathrm{NP}$

G :



$(G, 8) \in \mathrm{VC}?$ ✓

$(G, 5) \in \mathrm{VC}?$ ✓

$(G, 4) \in \mathrm{VC}?$ ✗

Today

- Both CLIQUE and VC are easily shown to be in NP
- But despite more than 50 years of effort, nobody was able to show they are in P
- There are many other problems with the same status
- Is there a reason we fail to show that these problems are in P?

Today

- Both CLIQUE and VC are easily shown to be in NP
- But despite more than 50 years of effort, nobody was able to show they are in P
- There are many other problems with the same status
- Is there a reason we fail to show that these problems are in P?

Yes, there is! They are some of the hardest problems in NP mentioned earlier

Intuition

If only one of these hardest problems is in P, then $\mathrm{P} = \mathrm{NP}$ (i.e., all of them are)

Agenda

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p\colon \mathbb{N} \to \mathbb{R}_{>0}$

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p\colon \mathbb{N} \to \mathbb{R}_{>0}$
- Fix some input $w = w_0 \cdots w_{n-1} \in \Sigma^*$ for M

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p\colon \mathbb{N} \to \mathbb{R}_{>0}$
- Fix some input $w = w_0 \cdots w_{n-1} \in \Sigma^*$ for M
- Then, every branch of the computation tree induced by M and w has length at most $p(n)$ (where $n = |w|$)

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p \colon \mathbb{N} \to \mathbb{R}_{>0}$
- Fix some input $w = w_0 \cdots w_{n-1} \in \Sigma^*$ for M
- Then, every branch of the computation tree induced by M and w has length at most $p(n)$ (where $n = |w|$)
- Also, M can visit at most $p(n) + 1$ cells before it terminates

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p\colon \mathbb{N} \to \mathbb{R}_{>0}$
- Fix some input $w = w_0 \cdots w_{n-1} \in \Sigma^*$ for M
- Then, every branch of the computation tree induced by M and w has length at most $p(n)$ (where $n = |w|$)
- Also, M can visit at most $p(n) + 1$ cells before it terminates
- For convenience, we encode a configuration $[q, \tau, \ell]$ by $\tau(1) \cdots \tau(\ell - 1)\ \ (\tau(\ell), q)\ \ \tau(\ell + 1) \cdots \tau(p(n) + 1) \in (\Gamma \cup (\Gamma \times Q))^{p(n)+1}$

Setup

We want to show $L \leq_p \mathrm{SAT}$ for every $L \in \mathrm{NP}$

- If $L \subseteq \Sigma^*$ is in NP, then there is a one-tape halting NTM M computing L with time complexity $p(n)$ for some polynomial $p\colon \mathbb{N} \to \mathbb{R}_{>0}$
- Fix some input $w = w_0 \cdots w_{n-1} \in \Sigma^*$ for M
- Then, every branch of the computation tree induced by M and w has length at most $p(n)$ (where $n = |w|$)
- Also, M can visit at most $p(n) + 1$ cells before it terminates
- For convenience, we encode a configuration $[q, \tau, \ell]$ by $\tau(1) \cdots \tau(\ell - 1) \;\; (\tau(\ell), q) \;\; \tau(\ell + 1) \cdots \tau(p(n) + 1) \in (\Gamma \cup (\Gamma \times Q))^{p(n)+1}$
- A single branch of the tree can be represented by a $(p(n) + 1) \times p(n)$ grid such that the entry at (i, j) represents (1) the content of the i-th tape cell in the j-th configuration, (2) whether the head is at that cell, and (3) the current state

Visualization

$p(n)$	$\$$									$\$$
$\vdots$	$\$$									$\$$
	$\$$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\$$
j	$\$$	X	1	$\#, q_\ell$	X	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
	$\$$	X	1	$\#$	$1, q_1^\#$	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
$\vdots$	$\$$	X	1	$\#, q_1$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
2	$\$$	X	$1, q_1$	$\#$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
1	$\$$	$1, s$	1	$\#$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
	0	1	$\cdots$					$\cdots$	$p(n)+1$	$p(n)+2$

i

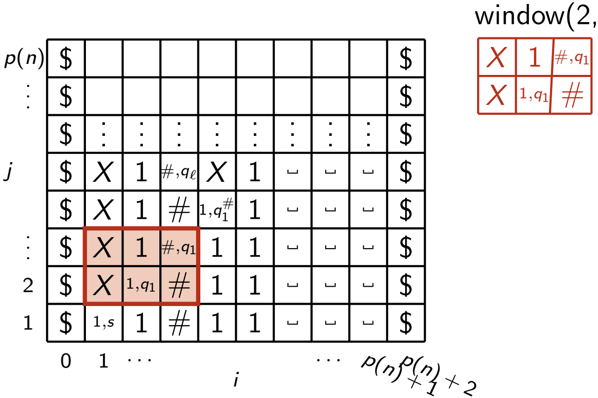
$p(n)$	\$									\$
$\vdots$	\$									\$
	\$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	\$
j	\$	X	1	$\#,q_\ell$	X	1	␣	␣	␣	\$
	\$	X	1	$\#$	$1,q_1^\#$	1	␣	␣	␣	\$
$\vdots$	\$	X	1	$\#,q_1$	1	1	␣	␣	␣	\$
2	\$	X	$1,q_1$	$\#$	1	1	␣	␣	␣	\$
1	\$	$1,s$	1	$\#$	1	1	␣	␣	␣	\$
	0	1	$\cdots$		i			$\cdots$	$p(n)+1$	$p(n)+2$

- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$

Visualization

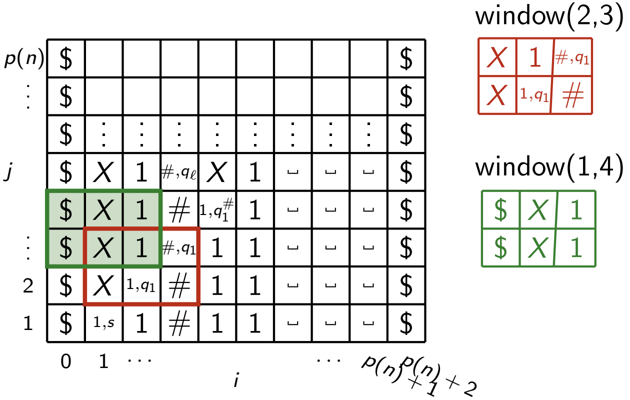
$p(n)$	$\$$									$\$$
$\vdots$	$\$$									$\$$
	$\$$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\$$
j	$\$$	X	1	$\#, q_\ell$	X	1	␣	␣	␣	$\$$
	$\$$	X	1	$\#$	$1, q_1^\#$	1	␣	␣	␣	$\$$
$\vdots$	$\$$	X	1	$\#, q_1$	1	1	␣	␣	␣	$\$$
2	$\$$	X	$1, q_1$	$\#$	1	1	␣	␣	␣	$\$$
1	$\$$	$1, s$	1	$\#$	1	1	␣	␣	␣	$\$$
	0	1	$\cdots$			i		$\cdots$	$p(n)+1$	$p(n)+2$

- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$
- Rely on locality: entry at (i, j) depends only on entries at $(i-1, j-1)$, $(i, j-1)$, and $(i+1, j-1)$: window of size 3×2



- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$
- Rely on locality: entry at (i,j) depends only on entries at $(i-1,j-1)$, $(i,j-1)$, and $(i+1,j-1)$: window of size 3×2

Visualization



- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$
- Rely on locality: entry at (i,j) depends only on entries at $(i-1, j-1)$, $(i, j-1)$, and $(i+1, j-1)$: window of size 3×2

Visualization

$p(n)$	$\$$									$\$$
$\vdots$	$\$$									$\$$
	$\$$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\$$
j	$\$$	X	1	$\#,q_\ell$	X	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
	$\$$	X	1	$\#$	$1,q_1^\#$	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
$\vdots$	$\$$	X	1	$\#,q_1$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
2	$\$$	X	$1,q_1$	$\#$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
1	$\$$	$1,s$	1	$\#$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\$$
	0	1	$\cdots$		i			$\cdots$	$p(n)+1$	$p(n)+2$

window(2,3)

X	1	$\#,q_1$
X	$1,q_1$	$\#$

window(1,4)

$\$$	X	1
$\$$	X	1

- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$
- Rely on locality: entry at (i,j) depends only on entries at $(i-1,j-1)$, $(i,j-1)$, and $(i+1,j-1)$: window of size 3×2
- How large is the grid?

Visualization

$p(n)$	\$									\$
$\vdots$	\$									\$
	\$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	\$
j	\$	X	1	$\#,q_\ell$	X	1	␣	␣	␣	\$
	\$	X	1	$\#$	$1,q_1^\#$	1	␣	␣	␣	\$
$\vdots$	\$	X	1	$\#,q_1$	1	1	␣	␣	␣	\$
2	\$	X	$1,q_1$	$\#$	1	1	␣	␣	␣	\$
1	\$	$1,s$	1	$\#$	1	1	␣	␣	␣	\$
	0	1	$\cdots$		i			$\cdots$	$p(n)+1$	$p(n)+2$

window(2,3)

X	1	$\#,q_1$
X	$1,q_1$	$\#$

window(1,4)

\$	X	1
\$	X	1

- Our goal: Given M and w, compute (in polynomial time) a formula $\varphi_{M,w}$ such that $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$
- Rely on locality: entry at (i,j) depends only on entries at $(i-1,j-1)$, $(i,j-1)$, and $(i+1,j-1)$: window of size 3×2
- How large is the grid? Polynomial in $p(n)$, and thus in n

Roadmap

$$w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$$

- $w \in L(M)$ is equivalent to the existence of a $(p(n)+3) \times p(n)$-grid encoding a branch of an accepting run
- We express the existence of such a grid by a formula:

  $\varphi_{M,w} = \varphi_{\text{setup}} \wedge \varphi_{\text{init}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}}$

- Define $\Gamma_{\$} = \Gamma \cup \{\$\}$[1]
- We use the following propositions (for $i \in \{0, 1, \ldots, p(n)+2\}$, $j \in \{1, 2, \ldots, p(n)\}$, $a \in \Gamma_{\$}$, and $q \in Q$)
  - $x_{i,j,a}$: grid entry at (i,j) is a, and
  - $x_{i,j,(a,q)}$: grid entry at (i,j) is (a,q)

[1] Here we assume that $\$ \notin \Gamma$ is a fresh tape symbol

Setup

$$\varphi_{\text{setup}} = \bigwedge_{i=0}^{p(n)+2} \bigwedge_{j=1}^{p(n)} \bigvee_{e \in E} \left(x_{i,j,e} \wedge \bigwedge_{e' \in E \setminus \{e\}} \neg x_{i,j,e'} \right)$$

where $E = \Gamma_\$ \cup (\Gamma_\$ \times Q)$ is the set of possible entries

Setup

$$\varphi_{\text{setup}} = \bigwedge_{i=0}^{p(n)+2} \bigwedge_{j=1}^{p(n)} \bigvee_{e \in E} \left(x_{i,j,e} \wedge \bigwedge_{e' \in E \setminus \{e\}} \neg x_{i,j,e'} \right)$$

where $E = \Gamma_\$ \cup (\Gamma_\$ \times Q)$ is the set of possible entries

This encodes:

- There is some entry e at (i, j),
- But no other entries $e' \neq e$

Setup

$$\varphi_{\text{setup}} = \bigwedge_{i=0}^{p(n)+2} \bigwedge_{j=1}^{p(n)} \bigvee_{e \in E} \left(x_{i,j,e} \wedge \bigwedge_{e' \in E \setminus \{e\}} \neg x_{i,j,e'} \right)$$

where $E = \Gamma_\$ \cup (\Gamma_\$ \times Q)$ is the set of possible entries

This encodes:

- There is some entry e at (i, j),
- But no other entries $e' \neq e$

Note

The size of φ_{setup} is $\mathcal{O}((p(n))^2 \cdot |E|^2)$ and therefore polynomial in $|\ulcorner M \urcorner| + n$

Init

$$\varphi_{\text{init}} = x_{0,1,\$} \wedge x_{1,1,(w_0,s)} \wedge \left(\bigwedge_{i=2}^{n} x_{i,1,w_{i-1}} \right) \wedge \left(\bigwedge_{i=n+1}^{p(n)+1} x_{i,1,\llcorner\lrcorner} \right) \wedge x_{p(n)+2,1,\$}$$

Init

$$\varphi_{\text{init}} = x_{0,1,\$} \wedge x_{1,1,(w_0,s)} \wedge \left(\bigwedge_{i=2}^{n} x_{i,1,w_{i-1}} \right) \wedge \left(\bigwedge_{i=n+1}^{p(n)+1} x_{i,1,\sqcup} \right) \wedge x_{p(n)+2,1,\$}$$

This encodes:

- The bottom row in the grid is filled by $\$w{\sqcup}^{p(n)-n+1}\$$

Init

$$\varphi_{\text{init}} = x_{0,1,\$} \wedge x_{1,1,(w_0,s)} \wedge \left(\bigwedge_{i=2}^{n} x_{i,1,w_{i-1}} \right) \wedge \left(\bigwedge_{i=n+1}^{p(n)+1} x_{i,1,\sqcup} \right) \wedge x_{p(n)+2,1,\$}$$

This encodes:

- The bottom row in the grid is filled by $\$w\sqcup^{p(n)-n+1}\$$

Note

The size of φ_{init} is $\mathcal{O}(p(n))$ and therefore polynomial in $|\ulcorner M \urcorner| + n$

Move

$$\varphi_{\text{move}} = \bigwedge_{i=1}^{p(n)+1} \bigwedge_{j=2}^{p(n)} \psi_{i,j} \quad \text{with}$$

$$\psi_{i,j} = \bigvee_{\begin{array}{|c|c|c|}\hline e_3 & e_4 & e_5 \\ \hline e_0 & e_1 & e_2 \\ \hline \end{array}} x_{i-1,j-1,e_0} \wedge x_{i,j-1,e_1} \wedge x_{i+1,j-1,e_2} \wedge x_{i-1,j,e_3} \wedge x_{i,j,e_4} \wedge x_{i+1,j,e_5}$$

where the disjunction ranges over all legal windows

Move

$$\varphi_{\mathsf{move}} = \bigwedge_{i=1}^{p(n)+1} \bigwedge_{j=2}^{p(n)} \psi_{i,j} \quad \text{with}$$

$$\psi_{i,j} = \bigvee_{\substack{\begin{array}{|c|c|c|}\hline e_3 & e_4 & e_5 \\ \hline e_0 & e_1 & e_2 \\ \hline \end{array}}} x_{i-1,j-1,e_0} \wedge x_{i,j-1,e_1} \wedge x_{i+1,j-1,e_2} \wedge x_{i-1,j,e_3} \wedge x_{i,j,e_4} \wedge x_{i+1,j,e_5}$$

where the disjunction ranges over all legal windows

This encodes:

- Every (3×2)-window is correctly filled
- Here, we assume $\delta(t, a)$ is defined for all $a \in \Gamma$ in order to extend accepting runs to length $p(n)$ to be able to fill the grid

Move

$$\varphi_{\text{move}} = \bigwedge_{i=1}^{p(n)+1} \bigwedge_{j=2}^{p(n)} \psi_{i,j} \quad \text{with}$$

$$\psi_{i,j} = \bigvee_{\begin{array}{|c|c|c|}\hline e_3 & e_4 & e_5 \\ \hline e_0 & e_1 & e_2 \\ \hline \end{array}} x_{i-1,j-1,e_0} \wedge x_{i,j-1,e_1} \wedge x_{i+1,j-1,e_2} \wedge x_{i-1,j,e_3} \wedge x_{i,j,e_4} \wedge x_{i+1,j,e_5}$$

where the disjunction ranges over all legal windows

This encodes:

- Every (3×2)-window is correctly filled
- Here, we assume $\delta(t, a)$ is defined for all $a \in \Gamma$ in order to extend accepting runs to length $p(n)$ to be able to fill the grid

Note

The size of φ_{move} is $\mathcal{O}((p(n))^2 \cdot |E|^6)$ and therefore polynomial in $|\ulcorner M \urcorner| + n$

Accept

$$\varphi_{\text{accept}} = \bigvee_{i=1}^{p(n)+1} \bigvee_{j=1}^{p(n)} \bigvee_{a\in\Gamma} x_{i,j,(a,t)}$$

Accept

$$\varphi_{\text{accept}} = \bigvee_{i=1}^{p(n)+1} \bigvee_{j=1}^{p(n)} \bigvee_{a \in \Gamma} x_{i,j,(a,t)}$$

This encodes:

- The accepting state appears somewhere in the grid

Accept

$$\varphi_{\text{accept}} = \bigvee_{i=1}^{p(n)+1} \bigvee_{j=1}^{p(n)} \bigvee_{a \in \Gamma} x_{i,j,(a,t)}$$

This encodes:

- The accepting state appears somewhere in the grid

Note

The size of φ_{accept} is $\mathcal{O}((p(n))^2)$ and therefore polynomial in $|\ulcorner M \urcorner| + n$

Wrapping Things Up

- The function mapping $\ulcorner M \urcorner$ and w to $\varphi_{M,w}$ is computable in polynomial time
- We have $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$:

Wrapping Things Up

- The function mapping $\ulcorner M \urcorner$ and w to $\varphi_{M,w}$ is computable in polynomial time
- We have $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$:
  - From an accepting branch in the computation tree of M on w, we can construct a grid by stacking the configurations on top of each other. This yields a satisfying assignment of $\varphi_{M,w}$

Wrapping Things Up

- The function mapping $\ulcorner M \urcorner$ and w to $\varphi_{M,w}$ is computable in polynomial time
- We have $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$:
  - From an accepting branch in the computation tree of M on w, we can construct a grid by stacking the configurations on top of each other. This yields a satisfying assignment of $\varphi_{M,w}$
  - A satisfying assignment of $\varphi_{M,w}$ can be turned into a grid, which starts with the initial configuration, each row encodes a successor configuration of the one below it, and it contains an accepting configuration. Hence, the grid represents an accepting branch in the computation tree of M on w

Wrapping Things Up

- The function mapping $\ulcorner M \urcorner$ and w to $\varphi_{M,w}$ is computable in polynomial time
- We have $w \in L(M) \Leftrightarrow \varphi_{M,w} \in \mathrm{SAT}$:
  - From an accepting branch in the computation tree of M on w, we can construct a grid by stacking the configurations on top of each other. This yields a satisfying assignment of $\varphi_{M,w}$
  - A satisfying assignment of $\varphi_{M,w}$ can be turned into a grid, which starts with the initial configuration, each row encodes a successor configuration of the one below it, and it contains an accepting configuration. Hence, the grid represents an accepting branch in the computation tree of M on w
- Hence, $L \leq_p \mathrm{SAT}$

Agenda

1. Some More Problems
2. SAT Can Simulate All Problems in NP
3. **NP-Hardness and NP-Completeness**
4. More NP-Complete Problems

Hardness and Completeness

We have seen that every language in NP can be reduced to SAT in polynomial time, and moreover that SAT is in NP

We have seen that every language in NP can be reduced to SAT in polynomial time, and moreover that SAT is in NP

Definition

Let L be a language

1. L is NP-hard if $L' \leq_p L$ for all $L' \in \mathrm{NP}$
2. L is NP-complete if L is NP-hard and in NP

Hardness and Completeness

We have seen that every language in NP can be reduced to SAT in polynomial time, and moreover that SAT is in NP

Definition

Let L be a language

1. L is NP-hard if $L' \leq_p L$ for all $L' \in \mathrm{NP}$
2. L is NP-complete if L is NP-hard and in NP

Corollary [Cook '61, Levin '63]

SAT is NP-complete

Quiz 1

If L is NP-hard, is L necessarily in NP?

Quiz 1

If L is NP-hard, is L necessarily in NP?

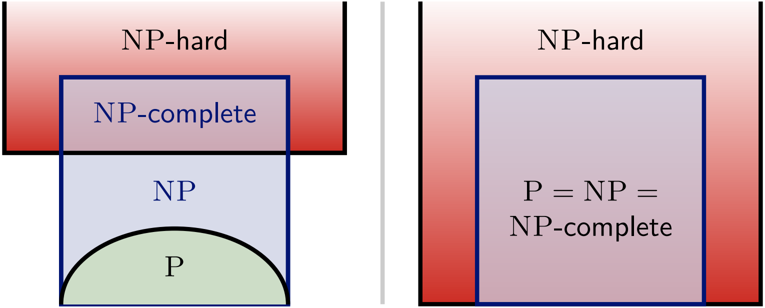
No, otherwise NP-hard and NP-complete would be identical

NP-hard really just means "very hard" – but there are still many different levels of "very hard"

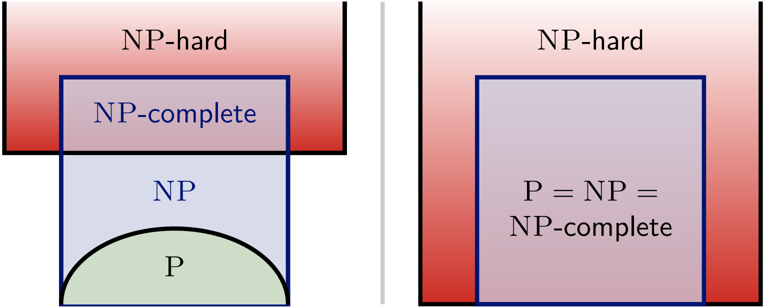
For instance, the halting problem is NP-hard (not shown here) but not in NP (because it is not even computable)

Possible Worlds

We are in one of the following two worlds:



We are in one of the following two worlds:



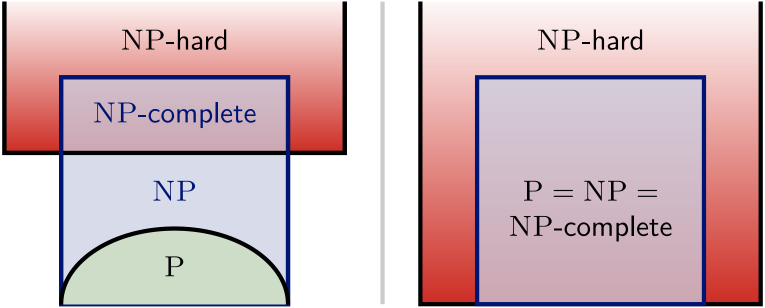
Theorem

Let L be NP-complete. If $L \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$

Proof

Exercise

We are in one of the following two worlds:



Theorem

Let L be NP*-complete. If* $L \in \mathrm{P}$, *then* $\mathrm{P} = \mathrm{NP}$

NP-complete problems are the hardest problems in NP:
a P-algorithm for L induces a P-algorithm for *all* $L' \in \mathrm{NP}$

P vs. NP

- In the 60 years since the Cook-Levin theorem, nobody has been able to show that any NP-complete problem is in P (and there are *many* candidates)
- Dually, nobody has shown that there is a problem in NP that is not in P

P vs. NP

- In the 60 years since the Cook-Levin theorem, nobody has been able to show that any NP-complete problem is in P (and there are *many* candidates)
- Dually, nobody has shown that there is a problem in NP that is not in P
- Most (but not all) computer scientists believe the latter to be the case

P vs. NP

- In the 60 years since the Cook-Levin theorem, nobody has been able to show that any NP-complete problem is in P (and there are *many* candidates)
- Dually, nobody has shown that there is a problem in NP that is not in P
- Most (but not all) computer scientists believe the latter to be the case
- But this has been hard to prove! In fact, we even have theorems that say such a proof must be hard (in some very technical sense: there are barriers to separations), e.g., a diagonalization proof cannot show $\mathrm{P} \neq \mathrm{NP}$

Exponential Time Hypothesis

Every halting DTM for SAT has exponential time complexity

Agenda

CNF-SAT

A Boolean formula φ is in conjunctive normal form (CNF) if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

CNF-SAT

A Boolean formula φ is in conjunctive normal form (CNF) if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Example

$(x \vee \neg y \vee z) \wedge (\neg x) \wedge (\neg y \vee \neg z)$ is in CNF

CNF-SAT

A Boolean formula φ is in conjunctive normal form (CNF) if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Example
$(x \vee \neg y \vee z) \wedge (\neg x) \wedge (\neg y \vee \neg z)$ is in CNF

Decision problem:

$$\mathrm{CNF\text{-}SAT} = \{\varphi \mid \varphi \text{ is in CNF and has a satisfying assignment}\}$$

CNF-SAT is NP-Complete

Recall from the last exercise sheet: $\mathrm{DNF\text{-}SAT}$ is in P

Theorem

$\mathrm{CNF\text{-}SAT}$ *is* NP*-complete*

CNF-SAT is NP-Complete

Recall from the last exercise sheet: $\mathrm{DNF\text{-}SAT}$ is in P

Theorem

$\mathrm{CNF\text{-}SAT}$ *is* NP*-complete*

Proof

$\varphi_{M,w}$ can be transformed into CNF in polynomial time (see exercises)

CNF-SAT is NP-Complete

Recall from the last exercise sheet: $\mathrm{DNF\text{-}SAT}$ is in P

Theorem

$\mathrm{CNF\text{-}SAT}$ *is* NP*-complete*

Proof

$\varphi_{M,w}$ can be transformed into CNF in polynomial time (see exercises)

Note

- Every Boolean formula can be turned into an equivalent one in DNF and into an equivalent one in CNF
- Thus, the structure of the formula has direct consequences for the complexity of the satisfiability problem

Quiz 2

Does the following algorithm for SAT run in polynomial time?

On input φ:

1. Transform φ into an equivalent formula φ' in DNF
2. Check whether φ' (and therefore φ) is satisfiable

Quiz 2

Does the following algorithm for SAT run in polynomial time?

On input φ:

1. Transform φ into an equivalent formula φ' in DNF
2. Check whether φ' (and therefore φ) is satisfiable

No, the first step is generally exponential

How to Prove Problems NP-Hard?

How can we prove that some L is NP-hard?

1. Show $L' \leq_p L$ for all $L' \in \mathrm{NP}$: **cumbersome**

How to Prove Problems NP-Hard?

How can we prove that some L is NP-hard?

1. Show $L' \leq_p L$ for all $L' \in \mathrm{NP}$: **cumbersome**
2. Reductions: much more elegant

How to Prove Problems NP-Hard?

How can we prove that some L is NP-hard?

1. Show $L' \leq_p L$ for all $L' \in \mathrm{NP}$: **cumbersome**
2. Reductions: much more elegant

Theorem

If $L' \leq_p L$ and L' is NP-hard, then L is also NP-hard

How to Prove Problems NP-Hard?

How can we prove that some L is NP-hard?

1. Show $L' \leq_p L$ for all $L' \in \mathrm{NP}$: **cumbersome**
2. Reductions: much more elegant

Theorem

If $L' \leq_p L$ and L' is NP-hard, then L is also NP-hard

Proof

Show that $\leq_p$ is transitive, i.e., $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ implies $L_1 \leq_p L_3$

3SAT is NP-complete

A Boolean formula φ is in 3CNF if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

3SAT is NP-complete

A Boolean formula φ is in 3CNF if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Example

$(x \vee \neg y \vee z) \wedge (\neg x \vee \neg x \vee \neg x) \wedge (\neg y \vee \neg z \vee \neg z)$ is in 3CNF

3SAT is NP-complete

A Boolean formula φ is in 3CNF if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Decision problem:

$$\mathrm{3SAT} = \{\varphi \mid \varphi \text{ is in 3CNF and has a satisfying assignment}\}$$

3SAT is NP-complete

A Boolean formula φ is in 3CNF if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Decision problem:

$$\mathrm{3SAT} = \{\varphi \mid \varphi \text{ is in 3CNF and has a satisfying assignment}\}$$

Theorem

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

3SAT is NP-complete

A Boolean formula φ is in 3CNF if it is of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$$

such that each $\ell_{i,j}$ is a literal, i.e., either a variable x or a negated variable $\neg x$

Decision problem:

$$\mathrm{3SAT} = \{\varphi \mid \varphi \text{ is in 3CNF and has a satisfying assignment}\}$$

Theorem

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

Proof

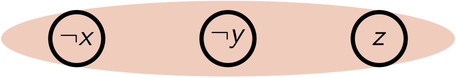
Exercise

Clique is NP-complete

- We show $3\mathrm{SAT} \leq_p \mathrm{CLIQUE}$
- First, we illustrate the reduction on an example

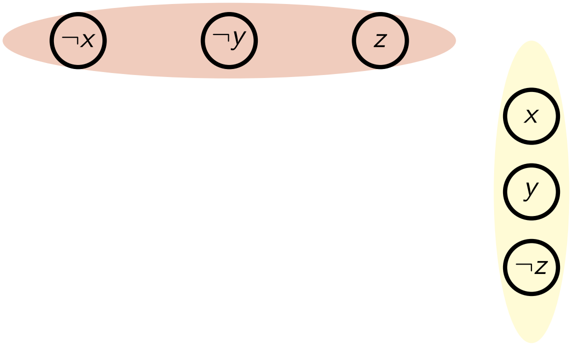
Clique is NP-complete

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



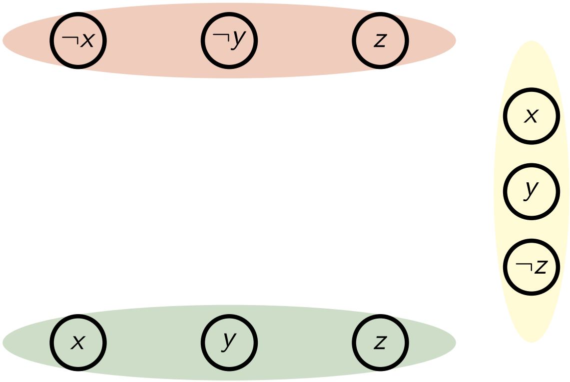
Clique is NP-complete

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



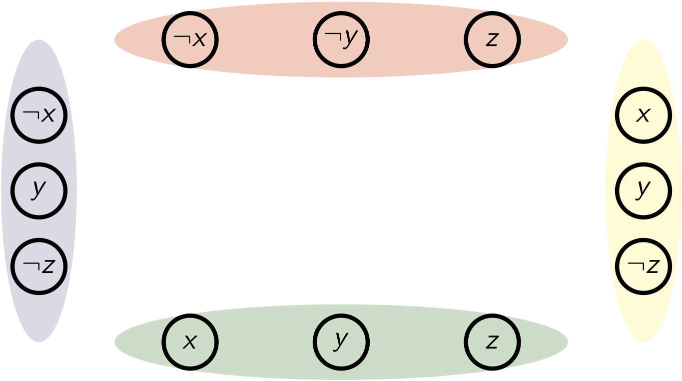
Clique is NP-complete

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



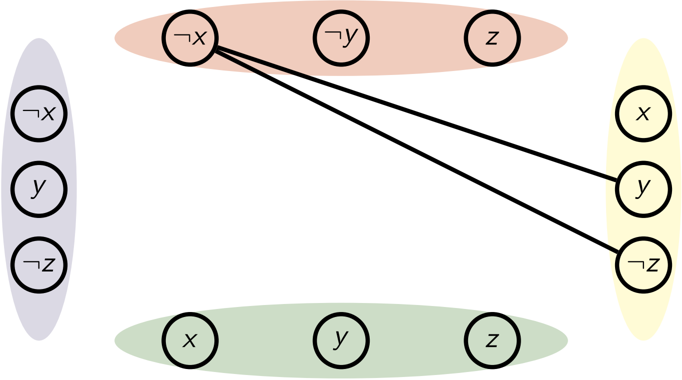
Clique is NP-complete

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



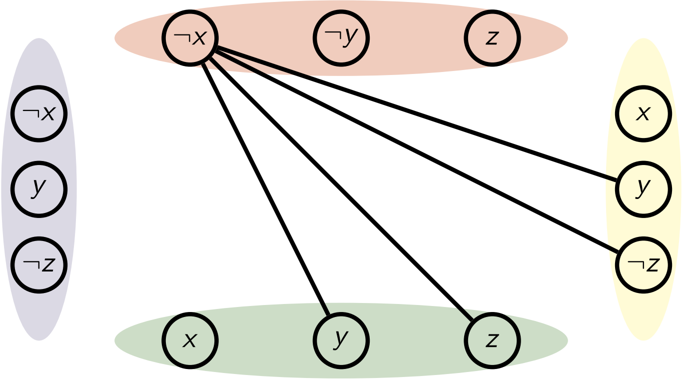
Clique is NP-complete

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



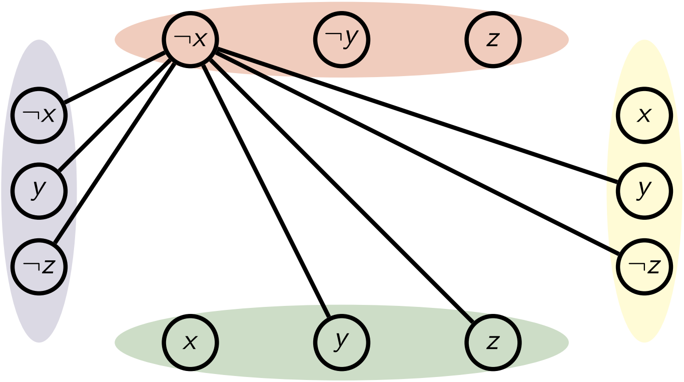
Clique is NP-complete

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

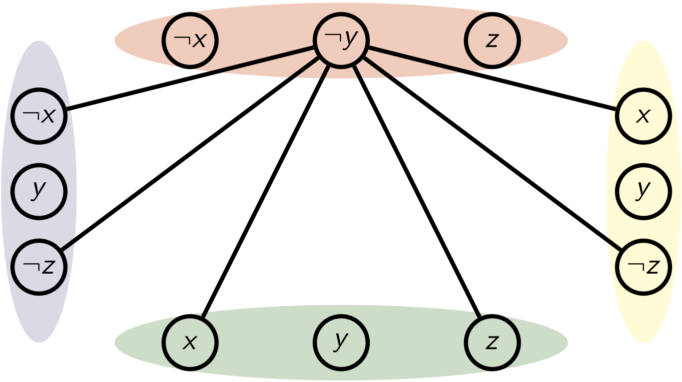


Clique is NP-complete

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

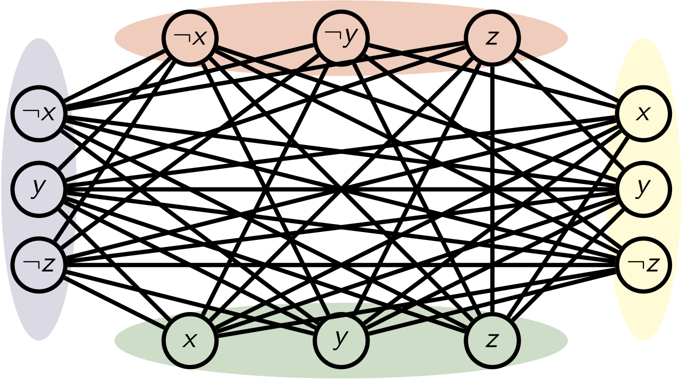


Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z)$



Clique is NP-complete

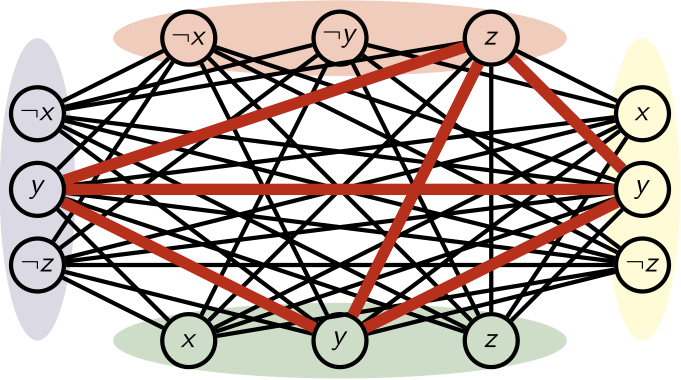
Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z)$



Clique is NP-complete

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

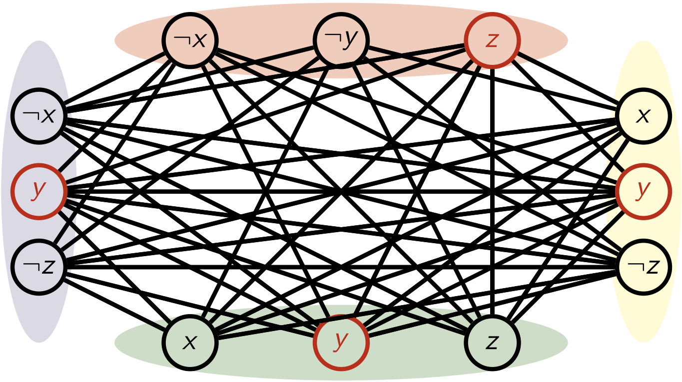
- A 4-clique contains exactly one vertex from each of the four clauses and no contradictory literals
- This can be extended to a satisfying assignment



Clique is NP-complete

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

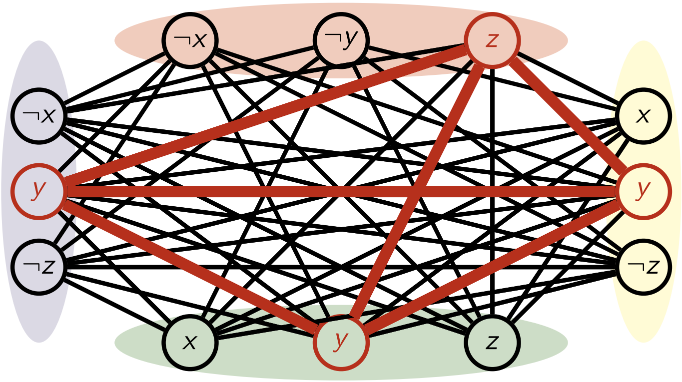
- Vice versa: A satisfying assignment satisfies some literal in each clause
- Picking one from each clause gives a 4-clique



Clique is NP-complete

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

- Vice versa: A satisfying assignment satisfies some literal in each clause
- Picking one from each clause gives a 4-clique



Clique is NP-complete

General construction for $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{i,j}$: Construct G_φ with

- One vertex for each literal $\ell_{i,j}$ ($3n$ nodes) and
- All edges but those between
  - Literals in the same clause and
  - Contradictory literals (e.g., x and $\neg x$)
- Then, we have $\varphi \in \mathrm{SAT} \Leftrightarrow (G_\varphi, n) \in \mathrm{CLIQUE}$

Vertex Cover is NP-complete

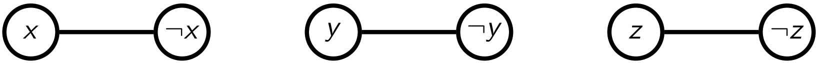
- We argue that $3\mathrm{SAT} \leq_p \mathrm{VC}$
- We only illustrate the reduction on an example

Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

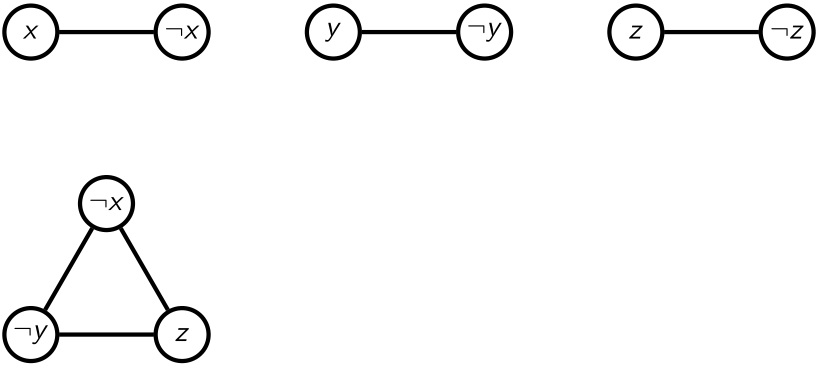
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



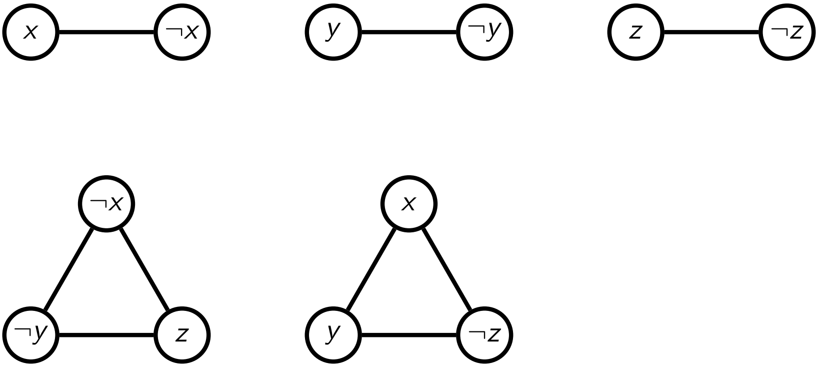
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



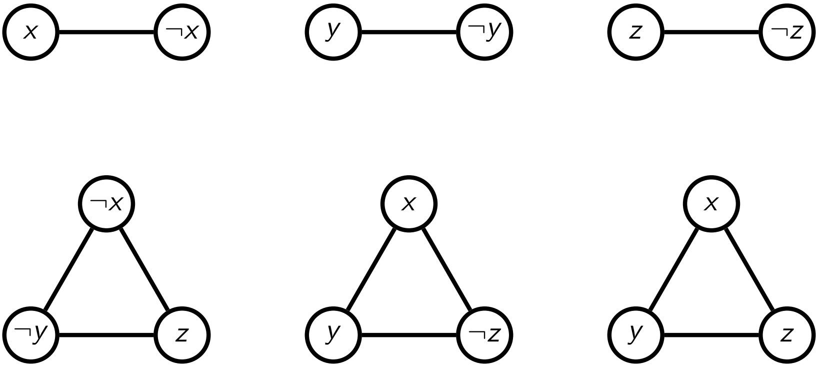
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



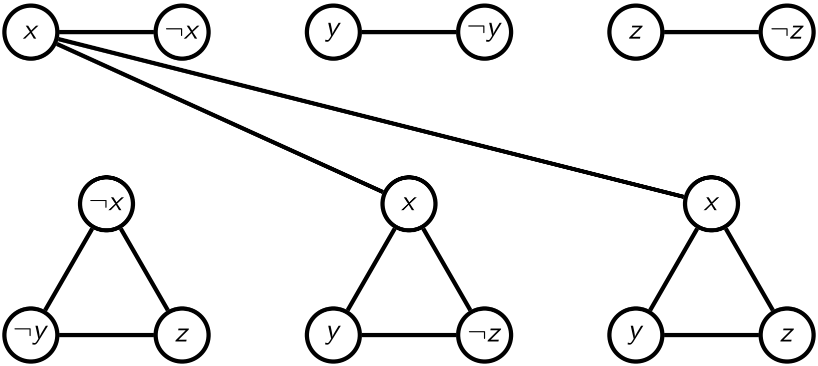
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



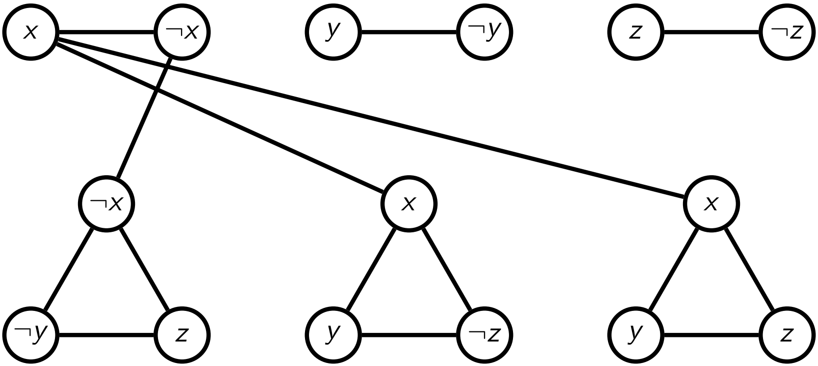
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



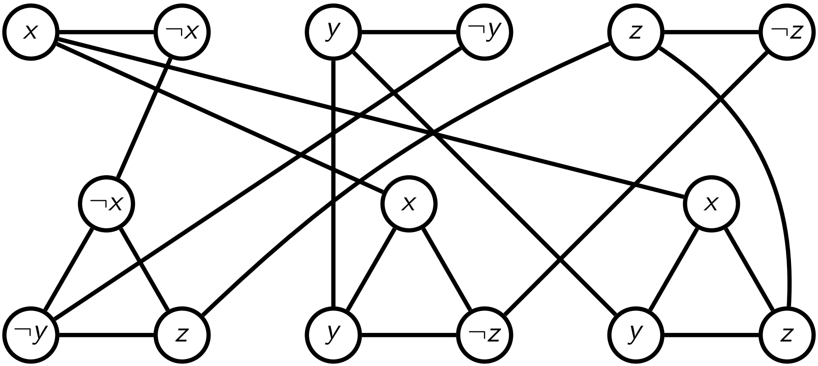
Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



Vertex Cover is NP-complete

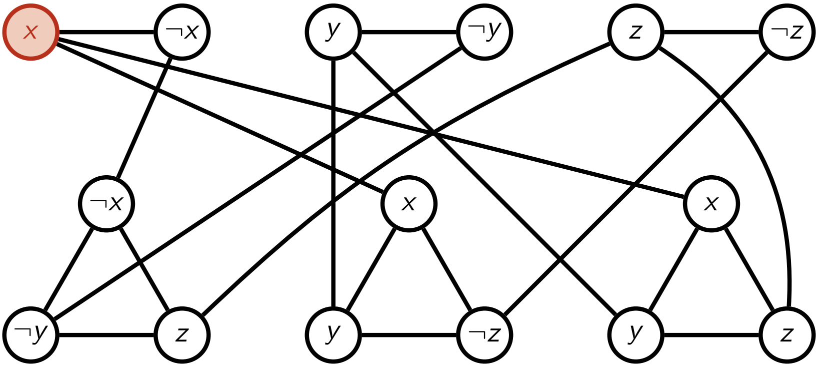
Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses



Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

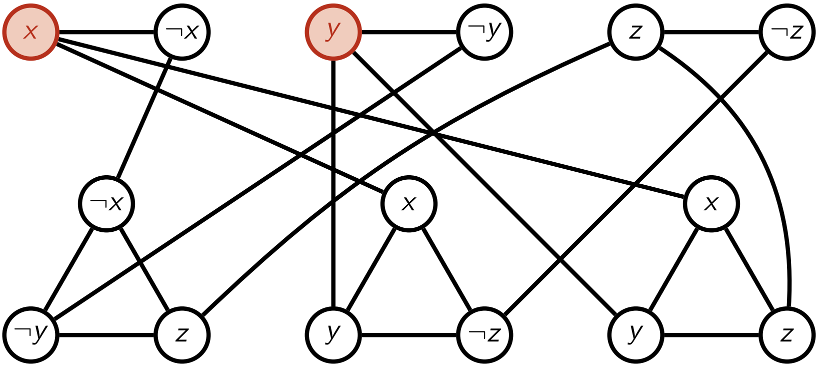
- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

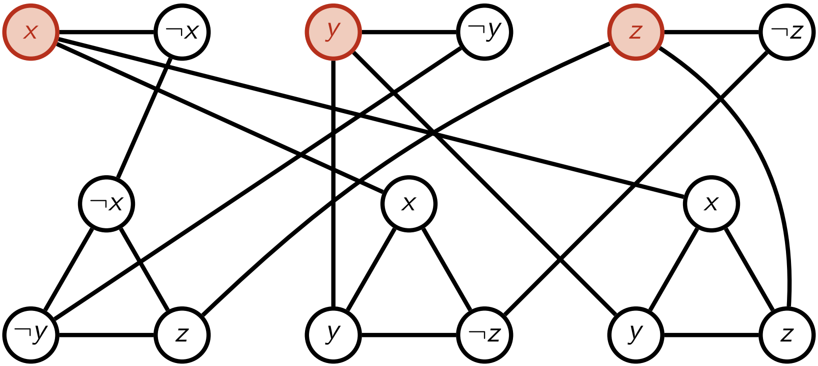
- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

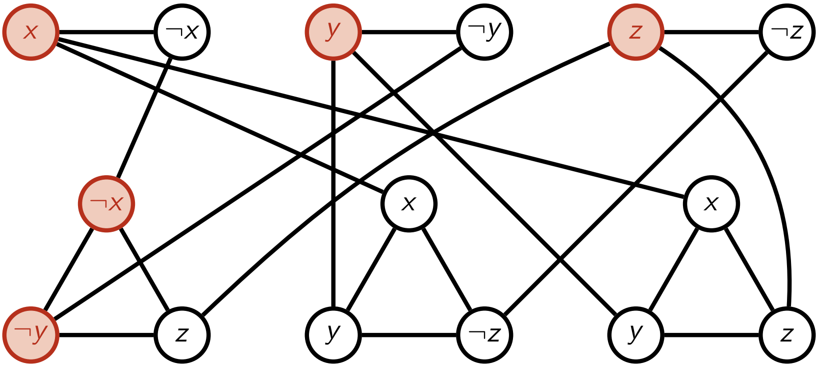
- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

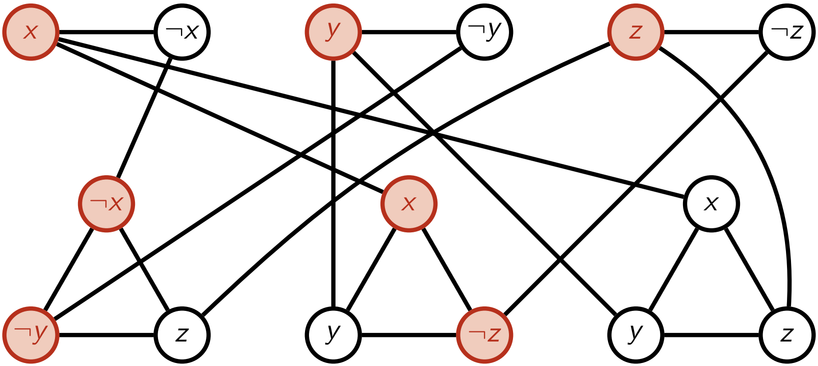
- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Vertex Cover is NP-complete

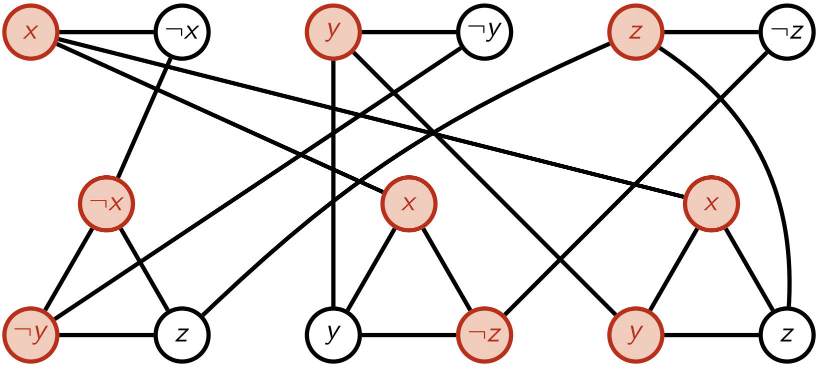
Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable



Vertex Cover is NP-complete

Let $\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ with $v = 3$ variables and $c = 3$ clauses

- The resulting graph has a $v + 2c = 9$-vertex cover if and only if φ is satisfiable
- Upper part ("variable pairs"):
  - Cover at least one vertex per pair ($\geq v$)
  - Intuition: Select an assignment for each variable
- Lower part ("clause triangles"):
  - Cover at least two vertices per triangle ($\geq 2c$)
  - Can omit one vertex that is taken care of by assignment
  - Intuition: Satisfy all clauses
- Since we can only cover $\leq v + 2c$ vertices, we have no slack

There is Much More

Many more problems are NP-complete

- **[Karp '72]**: 21 NP-complete problems, among them SAT, $\mathrm{3SAT}$, CLIQUE, VC, Hamiltonian path...
- **[Garey & Johnson '79]**: a book with 300 NP-hard problems
- More modern list: `https://en.wikipedia.org/wiki/List_of_NP-complete_problems`
- Some of these are very relevant in practice

Conclusion

We have seen

- The hardest problems in NP, the NP-complete ones
- Many useful problems are NP-complete
- If we can find a polynomial-time algorithm for one such problem, then we have one for all problems in NP, i.e., $\mathrm{P} = \mathrm{NP}$
- Researchers have tried to resolve the P vs. NP question for more than 60 years, without success
- Note: Public-key cryptography depends on the assumption that some problems cannot be solved efficiently

Reading

- Section 3.5 of "Computability and Complexity"
- Unfortunately, the book uses a completely different approach to NP-completeness