# Complexity Theory

## Lecture 6: NP-Complete Problems

**Tom Gur**

http://www.cl.cam.ac.uk/teaching/2324/Complexity

# Preface: What do professors do all day?

# Recap

- $\mathcal{P}$ captures polynomial-time computation.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.
- Cook-Levin Theorem: SAT is $\mathcal{NP}$-complete.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.
- Cook-Levin Theorem: SAT is $\mathcal{NP}$-complete.
- In fact, so is CNF-SAT.

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.
- Cook-Levin Theorem: SAT is $\mathcal{NP}$-complete.
- In fact, so is CNF-SAT.
- And CNF-SAT is reducible to 3SAT:

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.
- Cook-Levin Theorem: SAT is $\mathcal{NP}$-complete.
- In fact, so is CNF-SAT.
- And CNF-SAT is reducible to 3SAT:

# Recap

- $\mathcal{P}$ captures polynomial-time computation.
- $\mathcal{NP}$ captures polynomial-time verification.
- A problem is $\mathcal{NP}$-hard if any language in $\mathcal{NP}$ is reducible to it.
- A problem is $\mathcal{NP}$-complete if it is: (1) $\mathcal{NP}$-hard, (2) in $\mathcal{NP}$.
- Cook-Levin Theorem: SAT is $\mathcal{NP}$-complete.
- In fact, so is CNF-SAT.
- And CNF-SAT is reducible to 3SAT:
  $(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee z_2) \wedge (\neg z_2 \vee x_4)$

# Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

# Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

If we show, for some problem $A$ in NP that

$$\text{3SAT} \leq_P A$$

# Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

If we show, for some problem $A$ in NP that

$$3\text{SAT} \leq_P A$$

Then

$$\text{SAT} \leq_P 3\text{SAT} \leq_P A$$

# Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

If we show, for some problem $A$ in NP that

$$\text{3SAT} \leq_P A$$

Then

$$\text{SAT} \leq_P \text{3SAT} \leq_P A$$

Hence $A$ is also NP-complete.

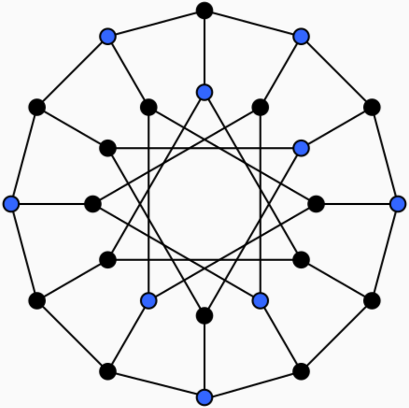
**Let’s see some reductions!**

# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.

# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.



# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.

The natural algorithmic problem is, given a graph, find the largest independent set.

# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.

The natural algorithmic problem is, given a graph, find the largest independent set.

To turn this *optimisation problem* into a *decision problem*, we define IS as:

> *The set of pairs $(G, K)$, where $G$ is a graph, and $K$ is an integer, such that $G$ contains an independent set with $K$ or more vertices.*

# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.

The natural algorithmic problem is, given a graph, find the largest independent set.

To turn this *optimisation problem* into a *decision problem*, we define IS as:

> *The set of pairs $(G, K)$, where $G$ is a graph, and $K$ is an integer, such that $G$ contains an independent set with $K$ or more vertices.*

# Independent Set

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is said to be an *independent set*, if there are no edges $(u, v)$ for $u, v \in X$.

The natural algorithmic problem is, given a graph, find the largest independent set.

To turn this *optimisation problem* into a *decision problem*, we define IS as:

> *The set of pairs $(G, K)$, where $G$ is a graph, and $K$ is an integer, such that $G$ contains an independent set with $K$ or more vertices.*

IS is clearly in NP. We now show it is NP-complete.

# Reduction

We can construct a reduction from 3SAT to IS.

# Reduction

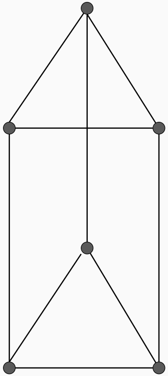
We can construct a reduction from 3SAT to IS.

A Boolean expression $\phi$ in 3CNF with $m$ clauses is mapped by the reduction to the pair $(G, m)$, where $G$ is the graph obtained from $\phi$ as follows:

> *$G$ contains $m$ triangles, one for each clause of $\phi$, with each node representing one of the literals in the clause.*
> *Additionally, there is an edge between two nodes in different triangles if they represent literals where one is the negation of the other.*

# Example

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$$

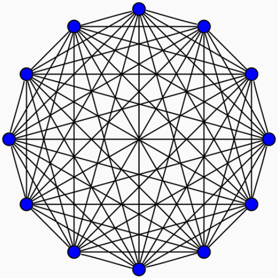


# Clique

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is called a *clique*, if for every $u, v \in X$, $(u, v)$ is an edge.

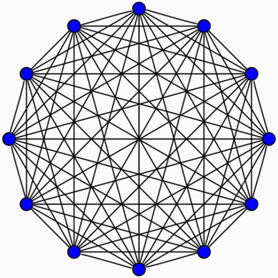
# Clique

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is called a *clique*, if for every $u, v \in X$, $(u, v)$ is an edge.



# Clique

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is called a *clique*, if for every $u, v \in X$, $(u, v)$ is an edge.



As with IS, we can define a decision problem:

CLIQUE is defined as:

*The set of pairs $(G, K)$, where $G$ is a graph, and $K$ is an integer, such that $G$ contains a clique with $K$ or more vertices.*

# Clique

CLIQUE is in NP by the algorithm which *guesses* a clique and then verifies it.

# Clique

CLIQUE is in NP by the algorithm which *guesses* a clique and then verifies it.

CLIQUE is NP-complete, since

IS $\leq_P$ CLIQUE

by the reduction that maps the pair $(G, K)$ to $(\bar{G}, K)$, where $\bar{G}$ is the complement graph of $G$.

# $k$-Colourability

A graph $G = (V, E)$ is $k$-colourable, if there is a function

$$\chi : V \to \{1, \dots, k\}$$

such that, for each $u, v \in V$, if $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$

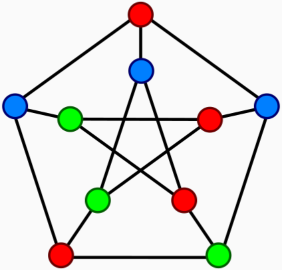
# $k$-Colourability

A graph $G = (V, E)$ is $k$-colourable, if there is a function

$$\chi : V \to \{1, \ldots, k\}$$

such that, for each $u, v \in V$, if $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$



# $k$-Colourability

A graph $G = (V, E)$ is $k$-colourable, if there is a function

$$\chi : V \to \{1, \dots, k\}$$

such that, for each $u, v \in V$, if $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$

This gives rise to a decision problem for each $k$.

# $k$-Colourability

A graph $G = (V, E)$ is $k$-colourable, if there is a function

$$\chi : V \to \{1, \ldots, k\}$$

such that, for each $u, v \in V$, if $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$

This gives rise to a decision problem for each $k$.

2-colourability is in P.

# $k$-Colourability

A graph $G = (V, E)$ is $k$-colourable, if there is a function

$$\chi : V \to \{1, \dots, k\}$$

such that, for each $u, v \in V$, if $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$

This gives rise to a decision problem for each $k$.

2-colourability is in P.

For all $k > 2$, $k$-colourability is NP-complete.

# 3-Colourability

3-Colourability is in NP, as we can *guess* a colouring and verify it.

# 3-Colourability

3-Colourability is in NP, as we can *guess* a colouring and verify it.

To show NP-completeness, we can construct a reduction from 3SAT to 3-Colourability.

# 3-Colourability

3-Colourability is in NP, as we can *guess* a colouring and verify it.

To show NP-completeness, we can construct a reduction from 3SAT to 3-Colourability.

For each variable $x$, we have two vertices $x$, $\bar{x}$ which are connected in a triangle with the vertex $a$ (common to all variables).

# 3-Colourability

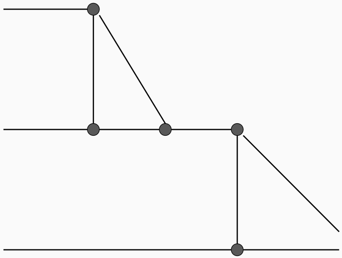
3-Colourability is in NP, as we can *guess* a colouring and verify it.

To show NP-completeness, we can construct a reduction from 3SAT to 3-Colourability.

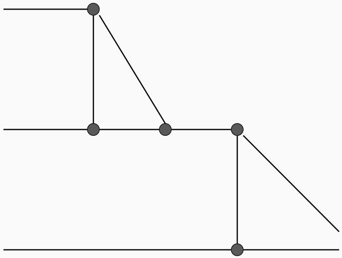
For each variable $x$, we have two vertices $x$, $\bar{x}$ which are connected in a triangle with the vertex $a$ (common to all variables).

In addition, for each clause containing the literals $l_1$, $l_2$ and $l_3$ we have a gadget.

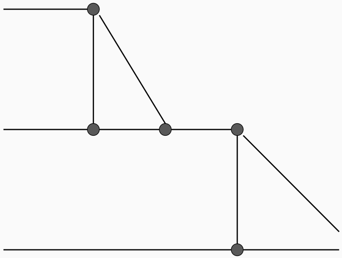
# Gadget



# Gadget



# Gadget



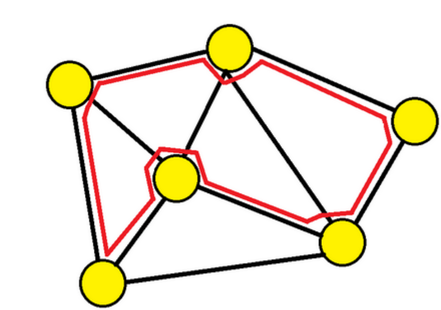
With a further edge from *a* to *b*.

# Hamiltonian Graphs

Given a graph $G = (V, E)$, a *Hamiltonian cycle* in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle *exactly once*.

# Hamiltonian Graphs

Given a graph $G = (V, E)$, a *Hamiltonian cycle* in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle *exactly once*.



# Hamiltonian Graphs

Given a graph $G = (V, E)$, a *Hamiltonian cycle* in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle *exactly once*.

A graph is called *Hamiltonian* if it contains a Hamiltonian cycle.

# Hamiltonian Graphs

Given a graph $G = (V, E)$, a *Hamiltonian cycle* in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle *exactly once*.

A graph is called *Hamiltonian* if it contains a Hamiltonian cycle.

The language HAM is the set of encodings of Hamiltonian graphs.

# Hamiltonian Cycle

We can construct a reduction from 3SAT to HAM

Essentially, this involves coding up a Boolean expression as a graph, so that every satisfying truth assignment to the expression corresponds to a Hamiltonian circuit of the graph.

# Hamiltonian Cycle

We can construct a reduction from 3SAT to HAM

Essentially, this involves coding up a Boolean expression as a graph, so that every satisfying truth assignment to the expression corresponds to a Hamiltonian circuit of the graph.

This reduction is much more intricate than the one for IND.

# Travelling Salesman

- $V$ — a set of nodes.

# Travelling Salesman

- $V$ — a set of nodes.
- $c : V \times V \to \mathbb{N}$ — a cost matrix.

# Travelling Salesman

- $V$ — a set of nodes.
- $c : V \times V \to \mathbb{N}$ — a cost matrix.

# Travelling Salesman

- $V$ — a set of nodes.
- $c : V \times V \to \mathbb{N}$ — a cost matrix.

Find an ordering $v_1, \ldots, v_n$ of $V$ minimising:

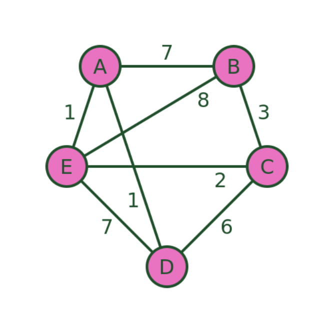
$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$

## Travelling Salesman

- $V$ — a set of nodes.
- $c : V \times V \to \mathbb{N}$ — a cost matrix.

Find an ordering $v_1, \ldots, v_n$ of $V$ minimising:

$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$



# Travelling Salesman

As with other optimisation problems, we can make a decision problem version of the Travelling Salesman problem.

# Travelling Salesman

As with other optimisation problems, we can make a decision problem version of the Travelling Salesman problem.

The problem TSP consists of the set of triples

$$(V, c : V \times V \to \mathbb{N}, t)$$

such that there is a tour of the set of vertices $V$, which under the cost matrix $c$, has cost $t$ or less.

# Reduction

There is a simple reduction from HAM to TSP, mapping a graph $(V, E)$ to the triple $(V, c : V \times V \to \mathbb{N}, n)$, where

$$c(u, v) = \begin{cases} 1 & (u, v) \in E \\ 2 & (u, v) \notin E \end{cases}$$

and $n$ is the size of $V$.

# Bonus: Randomness and BPP