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

- $P$ captures polynomial-time computation.

- $P$ captures polynomial-time computation.

- $P$ captures polynomial-time verification.

- A problem is $NP$-hard if any language in $NP$ is reducible to it.

- A problem is $NP$-complete if it is: (1) $NP$-hard, (2) in $NP$.

- Cook-Levin Theorem: $SAT$ is $NP$-complete.

- In fact, so is $CNF-SAT$.

- And $CNF-SAT$ is reducible to $3SAT$:

  $x_1 \lor x_2 \lor x_3 \lor x_4 \rightarrow (x_1 \lor \neg z_1 \lor x_2) \land (\neg z_1 \lor x_3 \lor z_2) \land (\neg z_2 \lor x_4)$
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 \( \mathcal{CNF-SAT} \).
- And \( \mathcal{CNF-SAT} \) is reducible to \( \mathcal{3SAT} \):
  \[
  \left( x_1 \lor x_2 \lor x_3 \lor x_4 \right) \rightarrow \left( x_1 \lor x_2 \lor z_1 \right) \land \left( \neg z_1 \lor x_3 \lor z_2 \right) \land \left( \neg z_2 \lor x_4 \right)
  \]
Recap

- \( P \) 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.

\[ (x_1 \lor x_2 \lor x_3 \lor x_4) \rightarrow (x_1 \lor x_2 \lor z_1) \land (\neg z_1 \lor x_3 \lor z_2) \land (\neg z_2 \lor x_4) \]
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:

\[
(x_1 \lor x_2 \lor x_3 \lor x_4) \rightarrow (x_1 \lor x_2 \lor z_1) \land (\neg z_1 \lor x_3 \lor z_2) \land (\neg z_2 \lor x_4)
\]
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 $3$SAT:
\( \mathcal{P} \) 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:
  \[
  (x_1 \lor x_2 \lor x_3 \lor x_4) \rightarrow (x_1 \lor x_2 \lor z_1) \land (\neg z_1 \lor x_3 \lor z_2) \land (\neg z_2 \lor x_4)
  \]
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 $L_1 \leq_P L_3$. 
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

$3\text{SAT} \leq_P A$
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

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

Then

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

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.
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 $\text{IS}$ 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.
We can construct a reduction from \textbf{3SAT} to \textbf{IS}.
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.
(x_1 \lor x_2 \lor \neg x_3) \land (x_3 \lor \neg x_2 \lor \neg x_1)
Clique

Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is called a \textit{clique}, if for every $u, v \in X$, $(u, v)$ is an edge.
Given a graph $G = (V, E)$, a subset $X \subseteq V$ of the vertices is called a \textit{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 is in NP by the algorithm which guesses a clique and then verifies it.
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 \(G\).
A graph $G = (V, E)$ is $k$-colourable, if there is a function

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

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

$$
\chi(u) \neq \chi(v)
$$
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 $(u, v) \in E$,

$$\chi(u) \neq \chi(v)$$
A graph $G = (V, E)$ is $k$-colourable, if there is a function

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

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

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

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

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

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 $P$. For all $k > 2$, $k$-colourability is $NP$-complete.
3-Colourability is in NP, as we can guess a colouring and verify it.
3-Colourability is in \textbf{NP}, as we can \textit{guess} a colouring and verify it.

To show \textbf{NP}-completeness, we can construct a reduction from \textbf{3SAT} to 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 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
With a further edge from $a$ to $b$. 
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*. 
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.
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 $\text{HAM}$ is the set of encodings of Hamiltonian graphs.
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.
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.
- $c : V \times V \rightarrow \mathbb{N}$ — a cost matrix.
Travelling Salesman

- \( V \) — a set of nodes.
- \( c : V \times V \rightarrow \mathbb{N} \) — a cost matrix.
Travelling Salesman

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

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

$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$
As with other optimisation problems, we can make a decision problem version of the Travelling Salesman problem.
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

\[(V, c : V \times V \rightarrow \mathbb{N}, t)\]

such that there is a tour of the set of vertices \(V\), which under the cost matrix \(c\), has cost \(t\) or less.
There is a simple reduction from HAM to TSP, mapping a graph \((V, E)\) to the triple \((V, c : V \times V \rightarrow \mathbb{N}, n)\), 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