NP-Complete
CSCI 338
Vertex Cover (VC)

Vertex Cover: Given graph $G = (V, E)$ and integer $k \leq |V|$, is there $V' \subseteq V$, with $|V'| \leq k$, such that each edge in $E$ contains an end point in $V'$?
Independent Set (IS)

Independent Set: Given a graph $G = (V, E)$ and integer $k \leq |V|$, is there a subset $V'$ of size $\geq k$, such that no two vertices $\in V'$ are adjacent?
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor x_2) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.
**SAT & 3SAT**

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.
\[ \phi = (x_1 \lor x_1 \lor x_2) \land (x_1 \lor \overline{x_2} \lor x_2) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.
\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor x_2) \land (x_1 \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.

(called conjunctive normal form – CNF)
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor x_2 \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\( \phi \) is a formula with clauses composed of Boolean variables connected by ORs, and clauses connected by ANDs.

Can you set the variables to **true** or **false** so that \( \phi \) evaluates to **true**?
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_2 \lor x_2) \]

\[ x_1 = false \]
\[ x_2 = true \]
$ SAT \ & \ 3SAT$

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

$$\downarrow \downarrow \downarrow$$

$$(F \lor F \lor T) \quad (T \lor F \lor F) \quad (T \lor T \lor T)$$

$x_1 = false$

$x_2 = true$
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (x_1 \lor x_2 \lor x_2) \land (x_1 \lor x_2 \lor x_2) \]

\[ (F \lor F \lor T) \quad (T \lor F \lor F) \quad (T \lor T \lor T) \]

\[ T \quad T \quad T \]

\[ x_1 = false \]
\[ x_2 = true \]
SAT & 3SAT

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]

\[ (F \lor F \lor T) \quad (T \lor F \lor F) \quad (T \lor T \lor T) \]

\[ T \quad T \quad T \]

\[ x_1 = false \quad x_2 = true \]
**SAT & 3SAT**

\[
\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)
\]

\[
(F \lor F \lor T) \quad (T \lor F \lor F) \quad (T \lor T \lor T)
\]

\[
x_1 = \text{false} \quad x_2 = \text{true}
\]

\[
SAT = \{\langle \phi \rangle: \phi \text{ is a satisfiable formula} \}
\]

\[
3SAT = \{\langle \phi \rangle: \phi \text{ is a satisfiable formula with 3 variables per clause} \}
\]
$P$ and $NP$
$P$ and $NP$

Stuff we can solve in polynomial time.
$P$ and $NP$

Stuff we can solve in polynomial time.
$P$ and $NP$

- $P$: Stuff we can solve in polynomial time.
- $NP$: Stuff we can verify solutions to in polynomial time.
$P$ versus $NP$
$NP$-Complete

$P \subseteq NP$

$NP$-Complete
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.
A solution to an *NP-Complete problem* can be used to solve any problem in *NP*, with just polynomial extra time.
A solution to an *NP*-Complete problem can be used to solve *any problem in NP*, with just polynomial extra time.
A solution to an \textit{NP}-Complete problem can be used to solve any problem in \textit{NP}, with just polynomial extra time.
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.
A solution to an \textit{NP-Complete} problem can be used to solve any problem in \textit{NP}, with just polynomial extra time.
A solution to an *NP*-Complete problem can be used to solve any problem in *NP*, with just polynomial extra time.

*NP*-Complete Problems:
- Vertex Cover
A solution to an \textit{NP-Complete problem} can be used to solve \textit{any problem in NP}, with just polynomial extra time.
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.

$NP$-Complete Problems:
- Vertex Cover
- Independent Set
- SAT
- 3-SAT
A solution to an \textit{NP-Complete} problem can be used to solve \textit{any problem in NP}, with just polynomial extra time.

\textit{NP-Complete} Problems:
- Vertex Cover
- Independent Set
- SAT
- 3-SAT

What if \exists polynomial time algorithm for \textit{Vertex Cover}?
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.

What if $\exists$ polynomial time algorithm for Vertex Cover?
- It could be used to solve any problem in $NP$ in polynomial time.

$NP$-Complete

$NP$-Complete Problems:
- Vertex Cover
- Independent Set
- SAT
- 3-SAT
A solution to an \textit{NP-Complete problem} can be used to solve \textit{any problem in NP}, with just polynomial extra time.

\textit{NP-Complete Problems:}
\begin{itemize}
  \item Vertex Cover
  \item Independent Set
  \item SAT
  \item 3-SAT
\end{itemize}

What if $\exists$ polynomial time algorithm for \textit{Vertex Cover}?
\begin{itemize}
  \item It could be used to solve \textit{any problem in NP} in polynomial time.
  \item $P = NP$.
A solution to an $\textit{NP-Complete}$ problem can be used to solve any problem in $\textit{NP}$, with just polynomial extra time.

What if $\exists$ polynomial time algorithm for $\textit{3-SAT}$?
A solution to an $\text{NP}$-Complete problem can be used to solve any problem in $\text{NP}$, with just polynomial extra time.

What if $\exists$ polynomial time algorithm for $\text{3-SAT}$?
- It could be used to solve any problem in $\text{NP}$ in polynomial time.

$\text{NP}$-Complete Problems:
- Vertex Cover
- Independent Set
- $\text{SAT}$
- $\text{3-SAT}$
A solution to an \textit{NP}-Complete problem can be used to solve \textit{any problem in NP}, with just polynomial extra time.

What if \(\exists\) polynomial time algorithm for 3-SAT?
- It could be used to solve \textit{any problem in NP} in polynomial time.
- \(P = NP\).
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.

$NP$-Complete Problems:
- Vertex Cover
- Independent Set
- SAT
- 3-SAT

What if $\exists$ polynomial time algorithm for 3-SAT?
- It could be used to solve any problem in $NP$ in polynomial time.
- $P = NP$.
$NP$-Complete

$NP$-Complete Problems:
- Vertex Cover
- Independent Set
- SAT
- 3-SAT
**NP-Complete**

**P Problems:**
- Shortest Path
- Searching
- Sorting

**NP Complete Problems:**
- Vertex Cover
- Independent Set
- SAT
- 3-SAT
Are there problems in $NP$, but not $P$ or $NP$-Complete?
Are there problems in $NP$, but not $P$ or $NP$-Complete?

- We don’t know.
Are there problems in \( NP \), but not \( P \) or \( NP \)-Complete?

- We don’t know. If so, \( P \neq NP \).
Are there problems in \( NP \), but not \( P \) or \( NP \)-Complete?

- We don’t know. If so, \( P \neq NP \).
- Suspected problems in \( NP \) but not \( P \) or \( NP \)-Complete:
Are there problems in \( NP \), but not \( P \) or \( NP \)-Complete?

- We don’t know. If so, \( P \neq NP \).
- Suspected problems in \( NP \) but not \( P \) or \( NP \)-Complete:
  - Graph Isomorphism.
Are there problems in $NP$, but not $P$ or $NP$-Complete?

- We don’t know. If so, $P \neq NP$.
- Suspected problems in $NP$ but not $P$ or $NP$-Complete:
  - Graph Isomorphism.
  - Integer Factorization.
A solution to an *NP*-Complete problem can be used to solve any problem in *NP*, with just polynomial extra time.
A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.

Too vague. Need formal definition.
Polynomial Time Reductions

A solution to an \textit{NP}-Complete problem can be used to solve any problem in \textit{NP}, with just polynomial extra time.

What do we mean by using the solution to a problem to solve another problem?
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

A solution to an $NP$-Complete problem can be used to solve any problem in $NP$, with just polynomial extra time.

What do we mean by using the solution to a problem to solve another problem?
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Language $A$ is **polynomial time reducible** to language $B$, written $A \leq_p B$, if a polynomial time function $f$ exists where,

$$\omega \in A \iff f(\omega) \in B$$
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm
Polynomial Time Reductions

Problem A Solver

A Input → B Input → B Solver → B Solution → A Solution

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm

List L → Sort Input → Sorting Algorithm → Sort Solution → Max Solution
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A Solver

A Input → B Input → B Solver → B Solution → A Solution

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm

List L → List L → Sorting Algorithm → Sorted List L’ → Max Solution
Polynomial Time Reductions

Problem A Solver

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm

return L'[0]
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Problem A: Find Max
Problem B: Sort

Find Max Algorithm

```
return L'[0]
```
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Our Responsibility
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Our Responsibility

To show A reduces to B:

• Show any instance of A can be translated to some instance of B.
• The solution to B can be translated back to a solution to A.
Polynomial Time Reductions

A reduces to B if A can be solved with a solver for B.

Our Responsibility

To show A reduces to B:
• Show *any* instance of A can be translated to *some* instance of B.
• The solution to B can be translated back to a solution to A.
\(NP\)-Complete

\(B\) is in \(NP\)-Complete if it satisfies two conditions:

1. \(B \in NP\).
2. For every \(A \in NP\), \(A \leq_P B\).
$NP$-Complete

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_{P} B$.

"Every problem in $NP$ can be solved by an algorithm for $B$ in polynomial extra time."
$NP$-Complete

$B$ is in $NP$-Complete if it satisfies two conditions:

1. $B \in NP$.
2. For every $A \in NP$, $A \leq_P B$.
NP-Complete

\[ B \text{ is in } NP-\text{Complete} \text{ if it satisfies two conditions:} \]

1. \( B \in NP \).
2. For every \( A \in NP \), \( A \leq_P B \).

Suppose \( B \) is an \( NP-C \) problem.
$NP$-Complete

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_P B$.

Suppose $B$ is an $NP$-C problem. Suppose $C$ is a problem in $NP$. 
Suppose $B$ is an $NP$-C problem. Suppose $C$ is a problem in $NP$. Suppose $B \leq_p C$.

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_p B$. 

$NP$-Complete

$P \subseteq NP$
\textit{NP}–Complete

\( B \) is in \textit{NP}–Complete if it satisfies two conditions:
1. \( B \in \textit{NP} \).
2. For every \( A \in \textit{NP} \), \( A \leq \_p B \).

Suppose \( B \) is an \textit{NP}–C problem.
Suppose \( C \) is a problem in \textit{NP}.
Suppose \( B \leq _p C \).
Prove \( C \) is an \textit{NP}–C problem:
\( B \) is in \( NP \)-Complete if it satisfies two conditions:

1. \( B \in NP \).
2. For every \( A \in NP \), \( A \leq_p B \).

Suppose \( B \) is an \( NP \)-C problem. Suppose \( C \) is a problem in \( NP \). Suppose \( B \leq_p C \).

Prove \( C \) is an \( NP \) – \( C \) problem:

\[ \forall A \in NP , A \leq_p B \]
NP-Complete

\[ P \subseteq NP \]

Suppose \( B \) is an \( NP \)-C problem. Suppose \( C \) is a problem in \( NP \). Suppose \( B \leq_P C \).

Prove \( C \) is an \( NP - C \) problem:
\[ \forall A \in NP, A \leq_P B \leq_P C \]
$NP$-Complete

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_p B$.

Suppose $B$ is an $NP$-C problem.
Suppose $C$ is a problem in $NP$.
Suppose $B \leq_p C$.
Prove $C$ is an $NP - C$ problem:
\[ \forall A \in NP, A \leq_p B \leq_p C \Rightarrow \forall A \in NP, A \leq_p C \]
$NP$-Complete

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_P B$.

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For some $A \in NP$-C, $A \leq_P B$. 
B is in NP-Complete if it satisfies two conditions:
1. $B \in \text{NP}$.
2. For some $A \in \text{NP}$, $A \leq_{p} B$. 

NP-Complete
Cook-Levin Theorem

Claim: $SAT \in NP$-Complete

Proof:
Cook-Levin Theorem

Claim: $SAT \in NP$-Complete

Proof:

$SAT = \{ \langle \phi \rangle : \phi \text{ is a satisfiable formula} \}$

E.g. $\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor x_2) \land (\overline{x_1} \lor x_2 \lor x_2)$
Cook-Levin Theorem

Claim: $SAT \in NP$-Complete

Proof: Elaborate...
$B$ is in $NP$-Complete if it satisfies two conditions:

1. $B \in NP$.
2. For every $A \in NP$, $A \leq_p B$.

$B$ is in $NP$-Complete if it satisfies two conditions:

1. $B \in NP$.
2. For some $A \in NP$-C, $A \leq_p B$. 
$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For every $A \in NP$, $A \leq_P B$.

$B$ is in $NP$-Complete if it satisfies two conditions:
1. $B \in NP$.
2. For some $A \in NP-C$, $A \leq_P B$. 

SAT
**NP-Complete**

How to show something ($B$) is in $NP$-Complete?

$B$ is in $NP$-Complete if it satisfies two conditions:

1. $B \in NP$.
2. For some $A \in NP-C$, $A \leq_p B$. 
**NP-Complete**

How to show something \((B)\) is in \(NP\)-Complete?

1. Show it is in \(NP\).

\(B\) is in \(NP\)-Complete if it satisfies two conditions:

1. \(B \in NP\).
2. For some \(A \in NP-C\), \(A \leq_p B\).
**NP-Complete**

How to show something \((B)\) is in \(NP\)-Complete?

1. Show it is in \(NP\).
2. Pick some known \(NP\)-Complete problem \(A\).

\[
\begin{align*}
B \text{ is in } NP\text{-Complete if it satisfies two conditions:} & \\
1. & B \in NP. \\
2. & \text{For some } A \in NP\text{-C, } A \leq_p B.
\end{align*}
\]
**NP-Complete**

How to show something \((B)\) is in \(NP\)-Complete?

1. Show it is in \(NP\).
2. Pick some known \(NP\)-Complete problem \(A\).
3. Show that a solver for \(B\) can solve \(A\) in polynomial extra time.

\(B\) is in \(NP\)-Complete if it satisfies two conditions:

1. \(B \in NP\).
2. For some \(A \in NP-C\), \(A \leq_p B\).