NP and Computational Intractability

T. M. Murali

March 17, 2014
Algorithm Design

- Patterns
  - Greed. \( O(n \log n) \) interval scheduling.
  - Divide-and-conquer. \( O(n \log n) \) closest pair of points.
  - Dynamic programming. \( O(n^2) \) edit distance.
  - Duality. \( O(n^3) \) maximum flow and minimum cuts.

\[ \text{NP vs. co-NP} \]
Algorithm Design

- Patterns
  - Greed. \( O(n \log n) \) interval scheduling.
  - Divide-and-conquer. \( O(n \log n) \) closest pair of points.
  - Dynamic programming. \( O(n^2) \) edit distance.
  - Duality. \( O(n^3) \) maximum flow and minimum cuts.
  - Reductions.
  - Local search.
  - Randomization.
Algorithm Design

- Patterns
  - Greed. \(O(n \log n)\) interval scheduling.
  - Divide-and-conquer. \(O(n \log n)\) closest pair of points.
  - Dynamic programming. \(O(n^2)\) edit distance.
  - Duality. \(O(n^3)\) maximum flow and minimum cuts.
  - Reductions.
  - Local search.
  - Randomization.

- “Anti-patterns”
  - NP-completeness. \(O(n^k)\) algorithm unlikely.
  - PSPACE-completeness. \(O(n^k)\) certification algorithm unlikely.
  - Undecidability. No algorithm possible.
Computational Tractability

- When is an algorithm an efficient solution to a problem?
Computational Tractability

- When is an algorithm an efficient solution to a problem? When its running time is polynomial in the size of the input.
Computational Tractability

- When is an algorithm an efficient solution to a problem? When its running time is polynomial in the size of the input.
- A problem is *computationally tractable* if it has a polynomial-time algorithm.
Computational Tractability

- When is an algorithm an efficient solution to a problem? When its running time is polynomial in the size of the input.
- A problem is *computationally tractable* if it has a polynomial-time algorithm.

<table>
<thead>
<tr>
<th>Polynomial time</th>
<th>Probably not</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>Matching</td>
<td>3-D matching</td>
</tr>
<tr>
<td>Minimum cut</td>
<td>Maximum cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar four-colour</td>
<td>Planar three-colour</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
</tr>
</tbody>
</table>
Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an $n$-by-$n$ board).
Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an $n$-by-$n$ board).
- However, classification is unclear for a very large number of discrete computational problems.
Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an $n$-by-$n$ board).
- However, classification is unclear for a very large number of discrete computational problems.
- We can prove that these problems are fundamentally equivalent and are manifestations of the same problem!
Polynomial-Time Reduction

- Goal is to express statements of the type “Problem $X$ is at least as hard as problem $Y$.”
  - Computing the maximum flow in a network is at least as hard as finding the minimum cut in a network.
- Use the notion of reductions.
- $Y$ is polynomial-time reducible to $X$ ($Y \leq_P X$)
Polynomial-Time Reduction

- Goal is to express statements of the type “Problem X is at least as hard as problem Y.”
  - Computing the maximum flow in a network is at least as hard as finding the minimum cut in a network.
- Use the notion of reductions.
- *Y is polynomial-time reducible to X* (*Y \leq^P X*) if any arbitrary instance of Y can be solved using a polynomial number of standard operations, plus a polynomial number of calls to a black box that solves problem X.
- *Y \leq^P X* implies that “X is at least as hard as Y.”
- Such reductions are *Cook reductions*. *Karp reductions* allow only one call to the black box that solves X.
Usefulness of Reductions

- Claim: If $Y \leq_{P} X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
Usefulness of Reductions

- Claim: If $Y \leq_P X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
- Contrapositive: If $Y \leq_P X$ and $Y$ cannot be solved in polynomial time, then $X$ cannot be solved in polynomial time.
- Informally: If $Y$ is hard, and we can show that $Y$ reduces to $X$, then the hardness “spreads” to $X$. 
Optimisation versus Decision Problems

- So far, we have developed algorithms that solve optimisation problems.
  - Compute the *largest* flow.
  - Find the *closest* pair of points.
  - Find the schedule with the *least* completion time.
Optimisation versus Decision Problems

- So far, we have developed algorithms that solve optimisation problems.
  - Compute the largest flow.
  - Find the closest pair of points.
  - Find the schedule with the least completion time.
- Now, we will focus on decision versions of problems, e.g., is there a flow with value at least $k$, for a given value of $k$?
- Decision problem: answer to every input is yes or no.

**Primes**

**INSTANCE:** A natural number $n$

**QUESTION:** Is $n$ prime?
Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an **independent set** if no two vertices in $S$ are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a **vertex cover** if every edge in $E$ is incident on at least one vertex in $S$. 

**Independent Set**

INSTANCE: Undirected graph $G$ and an integer $k$

QUESTION: Does $G$ contain an independent set of size $\geq k$?

**Vertex cover**

INSTANCE: Undirected graph $G$ and an integer $l$

QUESTION: Does $G$ contain a vertex cover of size $\leq l$?
Independent Set and Vertex Cover

Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in $S$ are connected by an edge.

Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in $E$ is incident on at least one vertex in $S$. 
Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an \textit{independent set} if no two vertices in $S$ are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a \textit{vertex cover} if every edge in $E$ is incident on at least one vertex in $S$.

\textbf{Independent Set}

\textbf{INSTANCE:} Undirected graph $G$ and an integer $k$

\textbf{QUESTION:} Does $G$ contain an independent set of size $\geq k$?

\textbf{Vertex Cover}

\textbf{INSTANCE:} Undirected graph $G$ and an integer $l$

\textbf{QUESTION:} Does $G$ contain a vertex cover of size $\leq l$?
Independent Set and Vertex Cover

Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in $S$ are connected by an edge.

Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in $E$ is incident on at least one vertex in $S$.

**Independent Set**

**INSTANCE:** Undirected graph $G$ and an integer $k$

**QUESTION:** Does $G$ contain an independent set of size $\geq k$?

**Vertex Cover**

**INSTANCE:** Undirected graph $G$ and an integer $l$

**QUESTION:** Does $G$ contain a vertex cover of size $\leq l$?

Demonstrate simple equivalence between these two problems.
**Independent Set and Vertex Cover**

Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an *independent set* if no two vertices in $S$ are connected by an edge. Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a *vertex cover* if every edge in $E$ is incident on at least one vertex in $S$.

**Independent Set**

**INSTANCE:** Undirected graph $G$ and an integer $k$

**QUESTION:** Does $G$ contain an independent set of size $\geq k$?

**Vertex cover**

**INSTANCE:** Undirected graph $G$ and an integer $l$

**QUESTION:** Does $G$ contain a vertex cover of size $\leq l$?

Demonstrate simple equivalence between these two problems.

Claim: $\text{Independent Set} \leq_P \text{Vertex Cover}$ and $\text{Vertex Cover} \leq_P \text{Independent Set}$. 
Strategy for Proving Indep. Set $\leq_p$ Vertex Cover

1. Start with an arbitrary instance of **Independent Set**: an undirected graph $G(V, E)$ and an integer $k$.

2. From $G(V, E)$ and $k$, create an instance of **Vertex Cover**: an undirected graph $G'(V', E')$ and an integer $l$.
   - $G'$ related to $G$ in some way.
   - $l$ can depend upon $k$ and size of $G$.

3. Prove that $G(V, E)$ has an independent set of size $\geq k$ iff $G'(V', E')$ has a vertex cover of size $\leq l$. 
Strategy for Proving Indep. Set \( \leq_P \) Vertex Cover

1. Start with an arbitrary instance of **INDEPENDENT SET**: an undirected graph \( G(V, E) \) and an integer \( k \).
2. From \( G(V, E) \) and \( k \), create an instance of **VERTEX COVER**: an undirected graph \( G'(V', E') \) and an integer \( l \).
   - \( G' \) related to \( G \) in some way.
   - \( l \) can depend upon \( k \) and size of \( G \).
3. Prove that \( G(V, E) \) has an independent set of size \( \geq k \) iff \( G'(V', E') \) has a vertex cover of size \( \leq l \).
   - Transformation and proof must be correct for all possible graphs \( G(V, E) \) and all possible values of \( k \).
   - Why is the proof an iff statement?
Strategy for Proving Indep. Set $\leq_P$ Vertex Cover

1. Start with an arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer $k$.
2. From $G(V, E)$ and $k$, create an instance of VERTEX COVER: an undirected graph $G'(V', E')$ and an integer $l$.
   - $G'$ related to $G$ in some way.
   - $l$ can depend upon $k$ and size of $G$.
3. Prove that $G(V, E)$ has an independent set of size $\geq k$ iff $G'(V', E')$ has a vertex cover of size $\leq l$.

- Transformation and proof must be correct for all possible graphs $G(V, E)$ and all possible values of $k$.
- Why is the proof an iff statement? In the reduction, we are using black box for VERTEX COVER to solve INDEPENDENT SET.
  (i) If there is an independent set size $\geq k$, we must be sure that there is a vertex cover of size $\leq l$, so that we know that the black box will find this vertex cover.
  (ii) If the black box finds a vertex cover of size $\leq l$, we must be sure we can construct an independent set of size $\geq k$ from this vertex cover.
Proof that Independent Set \( \leq_p \) Vertex Cover

1. Arbitrary instance of **Indepependent Set**: an undirected graph \( G(V, E) \) and an integer \( k \).
2. Let \( |V| = n \).
3. Create an instance of **Vertex Cover**: same undirected graph \( G(V, E) \) and integer \( n - k \).
Proof that Independent Set $\leq_p$ Vertex Cover

1. Arbitrary instance of **INDEPENDENT SET**: an undirected graph $G(V, E)$ and an integer $k$.

2. Let $|V| = n$.

3. Create an instance of **VERTEX COVER**: same undirected graph $G(V, E)$ and integer $n - k$.

4. Claim: $G(V, E)$ has an independent set of size $\geq k$ iff $G(V, E)$ has a vertex cover of size $\leq n - k$.

   **Proof**: $S$ is an independent set in $G$ iff $V - S$ is a vertex cover in $G$. 

T. M. Murali March 17, 2014 NP and Computational Intractability
Proof that Independent Set \( \leq_p \) Vertex Cover

1. Arbitrary instance of **INDEPENDENT SET**: an undirected graph \( G(V, E) \) and an integer \( k \).
2. Let \( |V| = n \).
3. Create an instance of **VERTEX COVER**: same undirected graph \( G(V, E) \) and integer \( n - k \).
4. Claim: \( G(V, E) \) has an independent set of size \( \geq k \) iff \( G(V, E) \) has a vertex cover of size \( \leq n - k \).

   Proof: \( S \) is an independent set in \( G \) iff \( V - S \) is a vertex cover in \( G \).

   ▶ Same idea proves that **VERTEX COVER \( \leq_p \) INDEPENDENT SET**
Vertex Cover and Set Cover

- **Independent Set** is a “packing” problem: pack as many vertices as possible, subject to constraints (the edges).
- **Vertex Cover** is a “covering” problem: cover all edges in the graph with as few vertices as possible.
- There are more general covering problems.

**Set Cover**

**INSTANCE:** A set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**QUESTION:** Is there a collection of $\leq k$ sets in the collection whose union is $U$?

Figure 8.2 An instance of the Set Cover Problem.
**Vertex Cover** \(\leq_P\) **Set Cover**

- **Input to** *Vertex Cover*: an undirected graph \(G(V, E)\) and an integer \(k\).
- Let \(|V| = n\).
- Create an instance \(\{U, \{S_1, S_2, \ldots S_n\}\}\) of **Set Cover** where
Vertex Cover $\leq_P$ Set Cover

$U = \{(x_1, x_2), (x_1, x_4), (x_2, x_3), (x_2, x_4), (x_2, x_7), (x_3, x_7), (x_4, x_5), (x_5, x_6), (x_5, x_7), (x_6, x_7)\}$

$S_1 = \{(x_1, x_2), (x_1, x_4)\}$

$S_2 = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_7)\}$

$S_3, S_4, S_5, S_6,$ and $S_7$ defined similarly.

▶ Input to **Vertex Cover**: an undirected graph $G(V, E)$ and an integer $k$.
▶ Let $|V| = n$.
▶ Create an instance $\{U, \{S_1, S_2, \ldots S_n\}\}$ of **Set Cover** where
  ▶ $U = E$,
  ▶ for each vertex $i \in V$, create a set $S_i \subseteq U$ of the edges incident on $i$. 

\[ S_1 = \{(x_1, x_2), (x_1, x_4)\} \]

\[ S_2 = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_7)\} \]

\[ S_3, S_4, S_5, S_6, \text{ and } S_7 \text{ defined similarly.} \]
**Vertex Cover $\leq_P$ Set Cover**

$U = \{(x_1, x_2), (x_1, x_4), (x_2, x_3), (x_2, x_4), (x_2, x_7), (x_3, x_7), (x_4, x_5), (x_5, x_6), (x_5, x_7), (x_6, x_7)\}$

$S_1 = \{(x_1, x_2), (x_1, x_4)\}$

$S_2 = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_7)\}$

$S_3, S_4, S_5, S_6, \text{ and } S_7$ defined similarly.

- **Input to Vertex Cover**: an undirected graph $G(V, E)$ and an integer $k$.
- **Let** $|V| = n$.
- **Create an instance** $\{U, \{S_1, S_2, \ldots, S_n\}\}$ of Set Cover where
  - $U = E$,
  - for each vertex $i \in V$, create a set $S_i \subseteq U$ of the edges incident on $i$.
- **Claim**: $U$ can be covered with fewer than $k$ subsets iff $G$ has a vertex cover with at most $k$ nodes.
- **Proof strategy**:
  1. If $G(V, E)$ has a vertex cover of size at most $k$, then $U$ can be covered with at most $k$ subsets.
  2. If $U$ can be covered with at most $k$ subsets, then $G(V, E)$ has a vertex cover of size at most $k$. 

T. M. Murali March 17, 2014 NP and Computational Intractability
Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Often used to specify problems, e.g., in AI.
Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Often used to specify problems, e.g., in AI.
- We are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of $n$ Boolean variables.
- Each variable can take the value 0 or 1.
- A term is a variable $x_i$ or its negation $\overline{x_i}$.
- A clause of length $l$ is a disjunction (or) of $l$ distinct terms $t_1 \lor t_2 \lor \cdots t_l$.
- A truth assignment for $X$ is a function $\nu : X \rightarrow \{0, 1\}$.
- An assignment satisfies a clause $C$ if at least one term in $C$ has the value 1 in the assignment.
- An assignment satisfies a collection of clauses $C_1, C_2, \ldots C_k$ if it causes each clause to take the value 1,
Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Often used to specify problems, e.g., in AI.
- We are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of $n$ Boolean variables.
- Each variable can take the value 0 or 1.
- A term is a variable $x_i$ or its negation $\overline{x_i}$.
- A clause of length $l$ is a disjunction (or) of $l$ distinct terms $t_1 \lor t_2 \lor \cdots t_l$.
- A truth assignment for $X$ is a function $\nu : X \to \{0, 1\}$.
- An assignment satisfies a clause $C$ if at least one term in $C$ has the value 1 in the assignment.
- An assignment satisfies a collection of clauses $C_1, C_2, \ldots C_k$ if it causes each clause to take the value 1, i.e., $C_1 \land C_2 \land \cdots C_k$ evaluates to 1.
  - $\nu$ is a satisfying assignment with respect to $C_1, C_2, \ldots C_k$.
  - set of clauses $C_1, C_2, \ldots C_k$ is satisfiable.
SAT and 3-SAT

Satisfiability Problem (SAT)

INSTANCE: A set of clauses $C_1, C_2, \ldots C_k$ over a set $X = \{x_1, x_2, \ldots x_n\}$ of $n$ variables.

QUESTION: Is there a satisfying truth assignment for $X$ with respect to $C$?
SAT and 3-SAT

3-Satisfiability Problem (SAT)

INSTANCE: A set of clauses $C_1, C_2, \ldots C_k$, each of length three, over a set $X = \{x_1, x_2, \ldots x_n\}$ of $n$ variables.

QUESTION: Is there a satisfying truth assignment for $X$ with respect to $C$?
SAT and 3-SAT

3-Satisfiability Problem (SAT)

INSTANCE: A set of clauses $C_1, C_2, \ldots C_k$, each of length three, over a set $X = \{x_1, x_2, \ldots x_n\}$ of $n$ variables.

QUESTION: Is there a satisfying truth assignment for $X$ with respect to $C$?

- SAT and 3-SAT are fundamental combinatorial search problems.
- We have to make $n$ independent decisions (the assignments for each variable) while satisfying a set of constraints.
- Satisfying each constraint in isolation is easy, but we have to make our decisions so that all constraints are satisfied simultaneously.
3-SAT and Independent Set

\[ C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3} \]
\[ C_2 = \overline{x_1} \lor x_2 \lor x_4 \]
\[ C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4} \]

- We want to prove \(3\text{-SAT} \leq_P \text{INDEPENDENT SET}\).
3-SAT and Independent Set

\[ C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3} \]
\[ C_2 = \overline{x_1} \lor x_2 \lor x_4 \]
\[ C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4} \]

1. Select \( x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1 \).

We want to prove \( 3\text{-SAT} \leq_P \text{INDEPENDENT SET} \).

Two ways to think about 3-SAT:

1. Make an independent 0/1 decision on each variable and succeed if we achieve one of three ways in which to satisfy each clause.
3-SAT and Independent Set

\[ C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3} \]
\[ C_2 = \overline{x_1} \lor x_2 \lor x_4 \]
\[ C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4} \]

1. Select \( x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1 \).
2. Choose one literal from each clause to evaluate to true.

We want to prove \( 3\text{-SAT} \leq_P \text{INDEPENDENT SET} \).

Two ways to think about 3-SAT:

1. Make an independent 0/1 decision on each variable and succeed if we achieve one of three ways in which to satisfy each clause.
2. Choose (at least) one term from each clause. Find a truth assignment that causes each chosen term to evaluate to 1. Ensure that no two terms selected conflict, e.g., select \( \overline{x_2} \) in \( C_1 \) and \( x_2 \) in \( C_2 \).
3-SAT and Independent Set

\[ C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3} \]
\[ C_2 = \overline{x_1} \lor x_2 \lor x_4 \]
\[ C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4} \]

1. Select \( x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1. \)
2. Choose one literal from each clause to evaluate to true.

Choices of selected literals imply \( x_1 = 0, x_2 = 0, x_4 = 1. \)

We want to prove \( 3\text{-SAT} \leq_p \text{INDEPENDENT SET}. \)

Two ways to think about 3-SAT:

1. Make an independent 0/1 decision on each variable and succeed if we achieve one of three ways in which to satisfy each clause.
2. Choose (at least) one term from each clause. Find a truth assignment that causes each chosen term to evaluate to 1. Ensure that no two terms selected conflict, e.g., select \( \overline{x_2} \) in \( C_1 \) and \( x_2 \) in \( C_2 \).
Proving 3-SAT $\leq_P$ Independent Set

$C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$

$C_2 = \overline{x_1} \lor x_2 \lor x_4$

$C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4}$

- We are given an instance of 3-SAT with $k$ clauses of length three over $n$ variables.
- Construct an instance of independent set: graph $G(V, E)$ with $3k$ nodes.
Proving $3\text{-SAT} \leq_p \text{Independent Set}$

We are given an instance of 3-SAT with $k$ clauses of length three over $n$ variables.

Construct an instance of independent set: graph $G(V, E)$ with $3k$ nodes.

- For each clause $C_i$, $1 \leq i \leq k$, add a triangle of three nodes $v_{i1}, v_{i2}, v_{i3}$ and three edges to $G$.
- Label each node $v_{ij}, 1 \leq j \leq 3$ with the $j$th term in $C_i$.
We are given an instance of 3-SAT with $k$ clauses of length three over $n$ variables.

Construct an instance of independent set: graph $G(V, E)$ with $3k$ nodes.

- For each clause $C_i$, $1 \leq i \leq k$, add a triangle of three nodes $v_{i1}, v_{i2}, v_{i3}$ and three edges to $G$.
- Label each node $v_{ij}$, $1 \leq j \leq 3$ with the $j$th term in $C_i$.
- Add an edge between each pair of nodes whose labels correspond to terms that conflict.
Proving $3$-SAT $\leq_P$ Independent Set

Claim: $3$-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$. 

\[
\begin{align*}
C_1 &= x_1 \lor \overline{x_2} \lor \overline{x_3} \\
C_2 &= \overline{x_1} \lor x_2 \lor x_4 \\
C_3 &= \overline{x_1} \lor x_3 \lor \overline{x_4}
\end{align*}
\]
Proving 3-SAT $\leq^P$ Independent Set

$C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$

$C_2 = \overline{x_1} \lor x_2 \lor x_4$

$C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4}$

Claim: 3-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: 

- For each variable $x_i$, only $x_i$ or $\overline{x_i}$ is the label of a node in $S$.
- If $x_i$ is the label of a node in $S$, set $x_i = 1$; else set $x_i = 0$.
- Why is each clause satisfied?
Proving $3$-SAT $\leq_P$ Independent Set

$C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$

$C_2 = \overline{x_1} \lor x_2 \lor x_4$

$C_3 = \overline{x_1} \lor x_3 \lor x_4$

Claim: 3-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: Each triangle in $G$ has at least one node whose label evaluates to 1. Set $S$ of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?
Proving 3-SAT $\leq_P$ Independent Set

$C_1 = x_1 \lor \overline{x}_2 \lor \overline{x}_3$

$C_2 = \overline{x}_1 \lor x_2 \lor x_4$

$C_3 = \overline{x}_1 \lor x_3 \lor x_4$

Claim: 3-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: Each triangle in $G$ has at least one node whose label evaluates to 1. Set $S$ of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?

Independent set $S$ of size $\geq k$ $\rightarrow$ satisfiable assignment:
Claim: 3-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: Each triangle in $G$ has at least one node whose label evaluates to 1. Set $S$ of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?

Independent set $S$ of size $\geq k$ $\rightarrow$ satisfiable assignment: the size of this set is $k$. How do we construct a satisfying truth assignment from the nodes in the independent set?
Claim: 3-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: Each triangle in $G$ has at least one node whose label evaluates to 1. Set $S$ of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?

Independent set $S$ of size $\geq k$ $\rightarrow$ satisfiable assignment: the size of this set is $k$. How do we construct a satisfying truth assignment from the nodes in the independent set?

For each variable $x_i$, only $x_i$ or $\overline{x_i}$ is the label of a node in $S$. Why?
Proving $3$-SAT $\leq_P$ Independent Set

Claim: $3$-SAT instance is satisfiable iff $G$ has an independent set of size at least $k$.

Satisfiable assignment $\rightarrow$ independent set of size $\geq k$: Each triangle in $G$ has at least one node whose label evaluates to 1. Set $S$ of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?

Independent set $S$ of size $\geq k \rightarrow$ satisfiable assignment: the size of this set is $k$. How do we construct a satisfying truth assignment from the nodes in the independent set?

- For each variable $x_i$, only $x_i$ or $\overline{x_i}$ is the label of a node in $S$. Why?
- If $x_i$ is the label of a node in $S$, set $x_i = 1$; else set $x_i = 0$.
- Why is each clause satisfied?
Transitivity of Reductions

Claim: If \( Z \leq_P Y \) and \( Y \leq_P X \), then \( Z \leq_P X \).
Transitivity of Reductions

- Claim: If $Z \leq_P Y$ and $Y \leq_P X$, then $Z \leq_P X$.
- We have shown

$$3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$$
Finding vs. Certifying

- Is it easy to check if a given set of vertices in an undirected graph forms an independent set of size at least $k$?
- Is it easy to check if a particular truth assignment satisfies a set of clauses?
Finding vs. Certifying

- Is it easy to check if a given set of vertices in an undirected graph forms an independent set of size at least $k$?
- Is it easy to check if a particular truth assignment satisfies a set of clauses?
- We draw a contrast between finding a solution and checking a solution (in polynomial time).
- Since we have not been able to develop efficient algorithms to solve many decision problems, let us turn our attention to whether we can check if a proposed solution is correct.
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes,
Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$. 
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.

  **Decision Problem**
  INPUT: a natural number $n$
  QUESTION: is $n$ a prime number?

  **Membership Problem**
  INPUT: a natural number $n$
  QUESTION: is $n$ in $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$?
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.

  **Decision Problem**
  **INPUT**: a natural number $n$
  **QUESTION**: is $n$ a prime number?

  **Membership Problem**
  **INPUT**: a natural number $n$
  **QUESTION**: is $n$ in $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$?

- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes}, \text{no}\}$.

- A *solves* the problem $X$ if for every input $s$, $A(s) = \text{yes}$ iff $s \in X$. 
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.

  **Decision Problem**
  
  **INPUT**: A natural number $n$
  
  **QUESTION**: Is $n$ a prime number?

  **Membership Problem**
  
  **INPUT**: A natural number $n$
  
  **QUESTION**: Is $n$ in $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$?

- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes}, \text{no}\}$.

- $A$ solves the problem $X$ if for every input $s$, $A(s) = \text{yes}$ iff $s \in X$.

- $A$ has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input $s$, $A$ terminates on $s$ in at most $O(p(|s|))$ steps.
  
  - There is an algorithm such that $p(|s|) = |s|^8$ for $\text{PRIMES}$ (Agarwal, Kayal, Saxena, 2002).
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.

  **Decision Problem**
  **INPUT**: a natural number $n$
  **QUESTION**: is $n$ a prime number?

  **Membership Problem**
  **INPUT**: a natural number $n$
  **QUESTION**: is $n$ in $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$?

- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes, no}\}$.
- $A$ solves the problem $X$ if for every input $s$, $A(s) = \text{yes}$ iff $s \in X$.
- $A$ has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input $s$, $A$ terminates on $s$ in at most $O(p(|s|))$ steps.
  - There is an algorithm such that $p(|s|) = |s|^8$ for $\text{PRIMES}$ (Agarwal, Kayal, Saxena, 2002).
- $\mathcal{P}$: set of problems $X$ for which there is a polynomial time algorithm.
Problems and Algorithms

- Equate a decision problem $X$ to the set of inputs for which the answer is yes, e.g., $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.
  
  **Decision Problem**
  
  **INPUT**: a natural number $n$
  
  **QUESTION**: is $n$ a prime number?

  **Membership Problem**

  **INPUT**: a natural number $n$
  
  **QUESTION**: is $n$ in $\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$?

- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes, no}\}$.

- $A$ solves the problem $X$ if for every input $s$, $A(s) = \text{yes}$ iff $s \in X$.

- $A$ has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input $s$, $A$ terminates on $s$ in at most $O(p(|s|))$ steps.
  
  - There is an algorithm such that $p(|s|) = |s|^8$ for $\text{PRIMES}$ (Agarwal, Kayal, Saxena, 2002).

- $\mathcal{P}$: set of problems $X$ for which there is a polynomial time algorithm.

- A decision problem $X$ is in $\mathcal{P}$ iff there is an algorithm $A$ with polynomial running time such that for all inputs $s$,
  
  - if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
  
  - if $X(s) = \text{no}$ then $A(s) = \text{no}$
Efficient Certification

- A “checking” algorithm for a decision problem $X$ has a different structure from an algorithm that solves $X$.
- Checking algorithm needs input $s$ as well as a separate “certificate” $t$ that contains evidence that $s \in X$.
- Checker for **Independent Set**: 

  - $t$ is a set of at least $k$ vertices; checker verifies that no pair of these vertices are connected by an edge.

An algorithm $B$ is an efficient certifier for a problem $X$ if

1. $B$ is a polynomial time algorithm that takes two inputs $s$ and $t$ and
2. for all inputs $s$ to $X$,
   - if $X(s) = \text{yes}$, then there is a proof $t$ such that $B(s, t) = \text{yes}$, and
   - if $X(s) = \text{no}$, then for all proofs $t$, $B(s, t) = \text{no}$.

- Certifier’s job is to take a candidate short proof ($t$) that $s \in X$ and check in polynomial time whether $t$ is a correct proof.
- Certifier does not care about how to find these proofs.
- False proofs cannot fool the certifier, so if $s \not\in X$ then there is no proof $t$ such that $B(s, t) = \text{yes}$. 

Efficient Certification

- A “checking” algorithm for a decision problem \( X \) has a different structure from an algorithm that solves \( X \).
- Checking algorithm needs input \( s \) as well as a separate “certificate” \( t \) that contains evidence that \( s \in X \).
- Checker for \textsc{Independent Set}: \( t \) is a set of at least \( k \) vertices; checker verifies that no pair of these vertices are connected by an edge.
Efficient Certification

▶ A “checking” algorithm for a decision problem $X$ has a different structure from an algorithm that solves $X$.

▶ Checking algorithm needs input $s$ as well as a separate “certificate” $t$ that contains evidence that $s \in X$.

▶ Checker for **Independent Set**: $t$ is a set of at least $k$ vertices; checker verifies that no pair of these vertices are connected by an edge.

▶ An algorithm $B$ is an **efficient certifier** for a problem $X$ if
  1. $B$ is a polynomial time algorithm that takes two inputs $s$ and $t$ and
  2. for all inputs $s$ to $X$,
     ▶ if $X(s) = \text{yes}$, then there is a proof $t$ such that $B(s, t) = \text{yes}$, and
     ▶ if $X(s) = \text{no}$, then for all proofs $t$, $B(s, t) = \text{no}$.
Efficient Certification

- A “checking” algorithm for a decision problem $X$ has a different structure from an algorithm that solves $X$.
- Checking algorithm needs input $s$ as well as a separate “certificate” $t$ that contains evidence that $s \in X$.
- Checker for **Independent Set**: $t$ is a set of at least $k$ vertices; checker verifies that no pair of these vertices are connected by an edge.
- An algorithm $B$ is an *efficient certifier* for a problem $X$ if
  1. $B$ is a polynomial time algorithm that takes two inputs $s$ and $t$ and
  2. for all inputs $s$ to $X$,
     - if $X(s) = \text{yes}$, then there is a proof $t$ such that $B(s, t) = \text{yes}$, and
     - if $X(s) = \text{no}$, then for all proofs $t$, $B(s, t) = \text{no}$.
- Certifier’s job is to take a candidate short proof ($t$) that $s \in X$ and check in polynomial time whether $t$ is a correct proof.
- Certifier does not care about how to find these proofs.
- False proofs cannot fool the certifier, so if $s \notin X$ then there is no proof $t$ such that $B(s, t) = \text{yes}$.
\[ \mathcal{NP} \]

- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \):
\[ \mathcal{NP} \]

- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \): \( t \) is a truth assignment; \( B \) evaluates the clauses with respect to the assignment.
\( \mathcal{NP} \)

- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \): \( t \) is a truth assignment; \( B \) evaluates the clauses with respect to the assignment.
- \text{Independent Set} \in \mathcal{NP}:
\( \text{NP} \)

- \( \text{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \text{NP} \): \( t \) is a truth assignment; \( B \) evaluates the clauses with respect to the assignment.
- \text{Independent Set} \in \text{NP}: \( t \) is a set of at least \( k \) vertices; \( B \) checks that no pair of these vertices are connected by an edge.
\( \mathsf{NP} \)

- \( \mathsf{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathsf{NP} \): \( t \) is a truth assignment; \( B \) evaluates the clauses with respect to the assignment.
- \( \text{Independent Set} \in \mathsf{NP} \): \( t \) is a set of at least \( k \) vertices; \( B \) checks that no pair of these vertices are connected by an edge.
- \( \text{Set Cover} \in \mathsf{NP} \):
\(\mathcal{NP}\)

- \(\mathcal{NP}\) is the set of all problems for which there exists an efficient certifier.
- \(3\text{-SAT} \in \mathcal{NP}\): \(t\) is a truth assignment; \(B\) evaluates the clauses with respect to the assignment.
- \(\text{Independent Set} \in \mathcal{NP}\): \(t\) is a set of at least \(k\) vertices; \(B\) checks that no pair of these vertices are connected by an edge.
- \(\text{Set Cover} \in \mathcal{NP}\): \(t\) is a list of \(k\) sets from the collection; \(B\) checks if their union is \(U\).
\( \mathcal{NP} \)

- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \): \( t \) is a truth assignment; \( B \) evaluates the clauses with respect to the assignment.
- \( \text{Independent Set} \in \mathcal{NP} \): \( t \) is a set of at least \( k \) vertices; \( B \) checks that no pair of these vertices are connected by an edge.
- \( \text{Set Cover} \in \mathcal{NP} \): \( t \) is a list of \( k \) sets from the collection; \( B \) checks if their union is \( U \).
- Proving that a problem is in \( \mathcal{NP} \) is usually quite easy!
Claim: $\mathcal{P} \subseteq \mathcal{NP}$. 

Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$? One of the major unsolved problems in computer science. $1M prize offered by Clay Mathematics Institute.
**P vs. NP**

- Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
  - If $X \in \mathcal{P}$, then there is a polynomial time algorithm $A$ that solves $X$. $B$ ignores $t$ and returns $A(s)$. Why is $B$ an efficient certifier?

---

**Diagram:**

- $\mathcal{P}$ is a subset of $\mathcal{NP}$. $\mathcal{P}$ is inside $\mathcal{NP}$.
\[ P \text{ vs. } NP \]

- **Claim:** \( P \subseteq NP \).
  - If \( X \in P \), then there is a polynomial time algorithm \( A \) that solves \( X \). \( B \) ignores \( t \) and returns \( A(s) \). Why is \( B \) an efficient certifier?
- Is \( P = NP \) or is \( NP - P \neq \emptyset \)?
**P vs. NP**

- Claim: $P \subseteq NP$.
  - If $X \in P$, then there is a polynomial time algorithm $A$ that solves $X$. $B$ ignores $t$ and returns $A(s)$. Why is $B$ an efficient certifier?
- Is $P = NP$ or is $NP - P \neq \emptyset$? One of the major unsolved problems in computer science. $1M prize offered by Clay Mathematics Institute.
What are the hardest problems in $\mathcal{NP}$?

A problem $X$ is $\mathcal{NP}$-Complete if:
1. $X \in \mathcal{NP}$
2. For every problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

A problem $X$ is $\mathcal{NP}$-Hard if:
1. For every problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

Claim: Suppose $X$ is $\mathcal{NP}$-Complete. Then $X \in \mathcal{P}$ iff $\mathcal{P} = \mathcal{NP}$.

Corollary: If there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no $\mathcal{NP}$-Complete problem can be solved in polynomial time.

Are there any $\mathcal{NP}$-Complete problems?

- What if two problems $X_1$ and $X_2$ in $\mathcal{NP}$ but there is no problem $X \in \mathcal{NP}$ where $X_1 \leq_P X$ and $X_2 \leq_P X$.

- Perhaps there is a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, each strictly harder than the previous one.
NP-Complete and NP-Hard Problems

What are the hardest problems in $\mathcal{NP}$?

A problem $X$ is **NP-Complete** if

(i) $X \in \mathcal{NP}$ and
(ii) for every problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

A problem $X$ is **NP-Hard** if

(i) for every problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

**Claim:** Suppose $X$ is NP-Complete. Then $X \in \mathcal{P}$ iff $\mathcal{P} = \mathcal{NP}$.

**Corollary:** If there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no NP-Complete problem can be solved in polynomial time.

**Are there any NP-Complete problems?**

1. What if two problems $X_1$ and $X_2$ in $\mathcal{NP}$ but there is no problem $X \in \mathcal{NP}$ where $X_1 \leq_P X$ and $X_2 \leq_P X$.
2. Perhaps there is a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, each strictly harder than the previous one.
**NP-Complete and NP-Hard Problems**

- What are the hardest problems in \( \mathcal{NP} \)?

A problem \( X \) is **NP-Complete** if

(i) \( X \in \mathcal{NP} \) and

(ii) for every problem \( Y \in \mathcal{NP} \), \( Y \leq_P X \).

A problem \( X \) is **NP-Hard** if

(i) for every problem \( Y \in \mathcal{NP} \), \( Y \leq_P X \).

- Claim: Suppose \( X \) is \( \mathcal{NP} \)-Complete. Then \( X \in \mathcal{P} \) iff \( \mathcal{P} = \mathcal{NP} \).

\[ \text{NP} \quad \text{NP-hard} \]

\[ \mathcal{P} \quad \mathcal{NPC} \]
**NP-Complete and NP-Hard Problems**

- What are the hardest problems in \( \mathsf{NP} \)?

A problem \( X \) is **NP-Complete** if

1. \( X \in \mathsf{NP} \) and
2. for every problem \( Y \in \mathsf{NP} \), \( Y \leq_{\mathsf{P}} X \).

A problem \( X \) is **NP-Hard** if

1. for every problem \( Y \in \mathsf{NP} \), \( Y \leq_{\mathsf{P}} X \).

- **Claim:** Suppose \( X \) is \( \mathsf{NP} \)-Complete. Then \( X \in \mathsf{P} \) iff \( \mathsf{P} = \mathsf{NP} \).

- **Corollary:** If there is any problem in \( \mathsf{NP} \) that cannot be solved in polynomial time, then no \( \mathsf{NP} \)-Complete problem can be solved in polynomial time.
**NP-Complete and NP-Hard Problems**

- What are the hardest problems in \( \text{NP} \)?

A problem \( X \) is **NP-Complete** if

1. \( X \in \text{NP} \)
2. For every problem \( Y \in \text{NP} \), \( Y \leq_P X \).

A problem \( X \) is **NP-Hard** if

1. For every problem \( Y \in \text{NP} \), \( Y \leq_P X \).

Claim: Suppose \( X \) is \( \text{NP} \)-Complete. Then \( X \in \mathcal{P} \) iff \( \mathcal{P} = \text{NP} \).

Corollary: If there is any problem in \( \text{NP} \) that cannot be solved in polynomial time, then no \( \text{NP} \)-Complete problem can be solved in polynomial time.

Are there any \( \text{NP} \)-Complete problems?

1. What if two problems \( X_1 \) and \( X_2 \) in \( \text{NP} \) but there is no problem \( X \in \text{NP} \) where \( X_1 \leq_P X \) and \( X_2 \leq_P X \).
2. Perhaps there is a sequence of problems \( X_1, X_2, X_3, \ldots \) in \( \text{NP} \), each strictly harder than the previous one.
Circuit Satisfiability

- **Cook-Levin Theorem**: \textsc{Circuit Satisfiability} is $\mathcal{NP}$-Complete.
Circuit Satisfiability

- **Cook-Levin Theorem**: Circuit Satisfiability is \( \mathcal{NP} \)-Complete.
- A *circuit* \( K \) is a labelled, directed acyclic graph such that
  1. the *sources* in \( K \) are labelled with constants (0 or 1) or the name of a distinct variable (the *inputs* to the circuit).
  2. every other node is labelled with one Boolean operator \( \wedge, \vee, \) or \( \neg \).
  3. a single node with no outgoing edges represents the *output* of \( K \).

Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.
Circuit Satisfiability

- **Cook-Levin Theorem:** Circuit Satisfiability is $NP$-Complete.
- A *circuit* $K$ is a labelled, directed acyclic graph such that
  1. the *sources* in $K$ are labelled with constants (0 or 1) or the name of a distinct variable (the *inputs* to the circuit).
  2. every other node is labelled with one Boolean operator $\land$, $\lor$, or $\neg$.
  3. a single node with no outgoing edges represents the *output* of $K$.

---

![Circuit Satisfiability Diagram](image)

**Circuit Satisfiability**

**INSTANCE:** A circuit $K$.

**QUESTION:** Is there a truth assignment to the inputs that causes the output to have value 1?

*Figure 8.4* A circuit with three inputs, two additional sources that have assigned truth values, and one output.
Asymmetry of Certification

Definition of efficient certification and $\mathcal{NP}$ is fundamentally asymmetric:

- An input string $s$ is a “yes” instance iff there exists a short string $t$ such that $B(s, t) = \text{yes}$.
- An input string $s$ is a “no” instance iff for all short strings $t$, $B(s, t) = \text{no}$. 

The definition of $\mathcal{NP}$ does not guarantee a short proof for “no” instances.
Asymmetry of Certification

Definition of efficient certification and $\mathcal{NP}$ is fundamentally asymmetric:

- An input string $s$ is a “yes” instance iff there exists a short string $t$ such that $B(s, t) = \text{yes}$.
- An input string $s$ is a “no” instance iff for all short strings $t$, $B(s, t) = \text{no}$.

The definition of $\mathcal{NP}$ does not guarantee a short proof for “no” instances.
co-$\mathcal{NP}$

- For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of strings $s$ such that $s \in \overline{X}$ iff $s \notin X$. 
For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of strings $s$ such that $s \in \overline{X}$ iff $s \notin X$.

If $X \in \mathcal{P}$,
co-\(NP\)

- For a decision problem \(X\), its *complementary problem* \(\overline{X}\) is the set of strings \(s\) such that \(s \in \overline{X}\) iff \(s \notin X\).
- If \(X \in P\), then \(\overline{X} \in P\).
For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of strings $s$ such that $s \in \overline{X}$ iff $s \notin X$.

- If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$?
For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of strings $s$ such that $s \in \overline{X}$ iff $s \notin X$.

- If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$? Unclear in general.
- A problem $X$ belongs to the class $\text{co-}\mathcal{NP}$ iff $\overline{X}$ belongs to $\mathcal{NP}$. 
**co-**\(\mathcal{NP}\)

- For a decision problem \(X\), its *complementary problem* \(\overline{X}\) is the set of strings \(s\) such that \(s \in \overline{X}\) iff \(s \notin X\).
- If \(X \in \mathcal{P}\), then \(\overline{X} \in \mathcal{P}\).
- If \(X \in \mathcal{NP}\), then is \(\overline{X} \in \mathcal{NP}\)? Unclear in general.
- A problem \(X\) belongs to the class *co-\(\mathcal{NP}\)* iff \(\overline{X}\) belongs to \(\mathcal{NP}\).

Open problem: Is \(\mathcal{NP} = \text{co-}\mathcal{NP}\)?
**co-NP**

- For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of strings $s$ such that $s \in \overline{X}$ iff $s \notin X$.
- If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$? Unclear in general.
- A problem $X$ belongs to the class $\text{co-NP}$ iff $\overline{X}$ belongs to $\mathcal{NP}$.

---

**Open problem:** Is $\mathcal{NP} = \text{co-NP}$?

**Claim:** If $\mathcal{NP} \neq \text{co-NP}$ then $\mathcal{P} \neq \mathcal{NP}$. 
Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- If a problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.
Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- If a problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.

- Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*. 

---

Example is the problem of determining if a flow network contains a flow of value at least $\nu$, for some given value of $\nu$.

- Yes: construct a flow of value at least $\nu$.
- No: demonstrate a cut with capacity less than $\nu$. 

---

**Claim:** $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$.

**Open problem:** Is $\mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}$?
Good Characterisations: the Class \( \mathcal{NP} \cap \text{co-} \mathcal{NP} \)

- If a problem belongs to both \( \mathcal{NP} \) and co-\( \mathcal{NP} \), then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.

- Problems in \( \mathcal{NP} \cap \text{co-} \mathcal{NP} \) have a good characterisation.

- Example is the problem of determining if a flow network contains a flow of value at least \( \nu \), for some given value of \( \nu \).
  - Yes: construct a flow of value at least \( \nu \).
  - No: demonstrate a cut with capacity less than \( \nu \).
Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- If a problem belongs to both $\mathcal{NP}$ and $\text{co-}\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.

- Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.

- Example is the problem of determining if a flow network contains a flow of value at least $\nu$, for some given value of $\nu$.
  - Yes: construct a flow of value at least $\nu$.
  - No: demonstrate a cut with capacity less than $\nu$.

Claim: $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$. 
Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- If a problem belongs to both $\mathcal{NP}$ and $\text{co-}\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.
- Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.
- Example is the problem of determining if a flow network contains a flow of value at least $\nu$, for some given value of $\nu$.
  - Yes: construct a flow of value at least $\nu$.
  - No: demonstrate a cut with capacity less than $\nu$.

Claim: $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$.

Open problem: Is $\mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}$?