NP and Computational Intractability

T. M. Murali

November 7, 12, 2018
Algorithm Design

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

Let's consider the algorithms by complexity:

MORE COMPLEX
- LEFTPAD
- QUICKSORT
- GIT
- MERGE
- SELF-DIVING
- GOOGLE
- SEARCH
- CAR
- BACKEND

SPRAWLING EXCEL SPREADSHEET
BUILT UP OVER 20 YEARS BY A CHURCH GROUP IN NEBRASKA TO COORDINATE THEIR SCHEDULING
Algorithm Design

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

Image segmentation $\leq_P$ Minimum $s-t$ cut
Algorithm Design

Patterns
- Greed. \(O(n \log n)\) interval scheduling.
- Divide-and-conquer. \(O(n \log n)\) closest pair of points.
- Dynamic programming. \(O(n^3)\) RNA folding.
- Duality.
- 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.

Image segmentation \(\leq_P\) Minimum \(s-t\) cut
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.

### Polynomial time
- Shortest path
- Matching
- Minimum cut
- 2-SAT
- Planar four-colour
- Bipartite vertex cover
- Primality testing

### Probably not
- Longest path
- 3-D matching
- Maximum cut
- 3-SAT
- Planar three-colour
- Vertex cover
- Factoring
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 $Y$.”
- 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 $Y$.”
- Use the notion of reductions.
- $Y$ is polynomial-time reducible to $X$ ($Y \leq_P X$) if any arbitrary instance (input) of $Y$ can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem $X$. 

Maximum Bipartite Matching $\leq_P$ Maximum $s$-$t$ Flow

Image Segmentation $\leq_P$ Minimum $s$-$t$ Cut

$Y \leq_P X$ implies that “$X$ is at least as hard as $Y$.”

Such reductions are Karp reductions. Cook reductions allow a polynomial number of calls to the black box that solves $X$. 

T. M. Murali November 7, 12, 2018 NP and Computational Intractability
**Polynomial-Time Reduction**

**Maximum Bipartite Matching $\leq_P$ Maximum $s-t$ Flow**

- **Algorithm for maximizing network flow**
- **Goal is to express statements of the type “Problem $X$ is at least as hard as problem $Y$.”**
- Use the notion of reductions.
- $Y$ is polynomial-time reducible to $X$ ($Y \leq_P X$) if any arbitrary instance (input) of $Y$ can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem $X$.

- **Maximum Bipartite Matching $\leq_P$ Maximum $s-t$ Flow**

**Polynomial-Time Reduction**
Polynomial-Time Reduction

- Goal is to express statements of the type “Problem X is at least as hard as problem Y.”
- Use the notion of reductions.
- \( Y \) is polynomial-time reducible to \( X \) (\( Y \leq_P X \)) if any arbitrary instance (input) of \( Y \) can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem \( X \).
Polynomial-Time Reduction

- Goal is to express statements of the type “Problem $X$ is at least as hard as problem $Y$.”
- Use the notion of reductions.
- $Y$ is polynomial-time reducible to $X$ ($Y \leq_p X$) if any arbitrary instance (input) of $Y$ can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem $X$.
  - Maximum Bipartite Matching $\leq_p$ Maximum $s$-$t$ Flow
  - Image Segmentation $\leq_p$ Minimum $s$-$t$ Cut
Polynomial-Time Reduction

- Goal is to express statements of the type “Problem $X$ is at least as hard as problem $Y$.”
- Use the notion of reductions.
- $Y$ is polynomial-time reducible to $X$ $(Y \leq_P X)$ if any arbitrary instance (input) of $Y$ can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem $X$.
  - Maximum Bipartite Matching $\leq_P$ Maximum $s$-$t$ Flow
  - Image Segmentation $\leq_P$ Minimum $s$-$t$ Cut
- $Y \leq_P X$ implies that “$X$ is at least as hard as $Y$.”
- Such reductions are Karp reductions. Cook reductions allow a polynomial number of calls 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 $X$. 
Reduction Strategies

- Simple equivalence.
- Special case to general case.
- Encoding with gadgets.
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 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 $S$. 
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 $\leq l$?
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 $\geq k$?

- Demonstrate simple equivalence between these two problems.
- **Claim:** Independent Set $\leq_P$ Vertex Cover and Vertex Cover $\leq_P$ Independent Set.
Strategy for Proving Indep. Set \( \leq_P \) Vertex Cover

\[ k = 3 \]

\( l = ? \)

Input graph for the vertex cover problem

Yes, there is an independent set of size at least 3

No, every independent set is of size 3 or less

Black box algorithm for solving vertex cover
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 $\leq l$. 

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?**
### Reason for Two-Way Proof

**Input graph for the vertex cover problem**

---

**Yes, there is an independent set of size at least 3**

**No, every independent set is of size 3 or less**

---

- Why is the proof an iff statement?
Reason for Two-Way Proof

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 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 $l = n - k$. 
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 $l = n - k$.
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 $G$. 
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 $l = n - k$.

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.

**Microbe Cover**

**INSTANCE:** A set $U$ of $n$ compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in $U$, and an integer $k$.

**QUESTION:** Is there a subset of $\leq k$ microbes that can together make all the compounds in $U$?

$M_1$:
- $c_1$
- $c_2$
- $c_4$
- $c_7$
- $c_6$
- $c_{10}$

$M_2$:
- $c_1$
- $c_4$
- $c_9$
- $c_5$
- $c_3$
- $c_6$
- $c_8$

$M_3$:
- $c_2$
- $c_9$
- $c_5$
- $c_{10}$
- $c_7$
- $c_3$

$M_4$:
- $c_1$
- $c_2$
- $c_4$
- $c_7$
- $c_6$
- $c_{10}$

$M_5$:
- $c_1$
- $c_4$
- $c_9$
- $c_5$
- $c_3$
- $c_6$
- $c_8$

$M_6$:
- $c_2$
- $c_9$
- $c_5$
- $c_{10}$
- $c_7$
- $c_3$

$n = 10, l = 6$
**Introduction Reductions**

**NP Complete**

**NP vs. co-NP**

**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.

**Microbe Cover**

**Instance**: A set $U$ of $n$ compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in $U$, and an integer $k$.

**Question**: Is there a subset of $\leq k$ microbes that can together make all the compounds in $U$?

$n = 10$, $l = 6$, $k = 3$
**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.

**Microbe Cover**

**Instance:** A set $U$ of $n$ compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in $U$, and an integer $k$.

**Question:** Is there a subset of $\leq k$ microbes that can together make all the compounds in $U$?

\[
\begin{align*}
M_1 & \quad c_1 \quad c_2 \\
& \quad c_4 \quad c_7 \\
M_2 & \quad c_1 \quad c_4 \quad c_9 \\
M_3 & \quad c_2 \quad c_9 \\
& \quad c_5 \quad c_{10} \\
M_4 & \quad c_6 \quad c_{10} \\
& \quad c_5 \quad c_3 \\
M_5 & \quad c_6 \quad c_8 \\
M_6 & \quad c_7 \quad c_3 \\
\end{align*}
\]

\[n = 10, l = 6, k = 3\]
Input to **Vertex Cover**: an undirected graph $G(V, E)$ and an integer $k$.

Let $|V| = l$.

Create an instance $\{U, \{M_1, M_2, \ldots, M_l\}\}$ of **Microbe Cover** where
**Vertex Cover \( \leq_p \) Microbe Cover**

- **Input to** **Vertex Cover**: an undirected graph \( G(V, E) \) and an integer \( k \).
- Let \( |V| = l \).
- Create an instance \( \{U, \{M_1, M_2, \ldots, M_l\}\} \) of **Microbe Cover** where
  - \( U = E \), i.e., each element of \( U \) is an edge of \( G \), and
  - for each node \( i \in V \), create a microbe \( M_i \) whose compounds are the set of edges incident on \( i \).
**Vertex Cover \( \leq_P \) Microbe Cover**

- **Input to Vertex Cover:** an undirected graph \( G(V, E) \) and an integer \( k \).
- Let \( |V| = l \).
- Create an instance \( \{ U, \{ M_1, M_2, \ldots, M_l \} \} \) of Microbe Cover where
  - \( U = E \), i.e., each element of \( U \) is an edge of \( G \), and
  - for each node \( i \in V \), create a microbe \( M_i \) whose compounds are the set of edges incident on \( i \).
- Claim: \( U \) can be covered with \( \leq k \) microbes iff \( G \) has a vertex cover with at \( \leq k \) nodes.
- **Proof strategy:**
  1. If \( G \) has a vertex cover of size \( \leq k \), then \( U \) can be covered with \( \leq k \) microbes.
  2. If \( U \) can be covered with \( \leq k \) microbes, then \( G \) has a vertex cover of size \( \leq k \).
Microbe Cover and Set Cover

**Microbe Cover**

**INSTANCE:** A set $U$ of $n$ compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in $U$, and an integer $k$.

**QUESTION:** Is there a subset of $\leq k$ microbes that can together make all the compounds in $U$?

- Purely combinatorial problem: a “microbe” is just a set of “compounds.”

**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 $U$?

$n = 10, l = 6$
Microbe Cover and Set Cover

**Microbe Cover**

**INSTANCE:** A set $U$ of $n$ compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in $U$, and an integer $k$.

**QUESTION:** Is there a subset of $\leq k$ microbes that can together make all the compounds in $U$?

- Purely combinatorial problem: a “microbe” is just a set of “compounds.”

**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 $U$?

$n = 10, l = 6$
Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of $n$ Boolean variables.
- Each variable can take the value 0 or 1.
- **Term**: a variable $x_i$ or its negation $\overline{x_i}$.
- **Clause of length $l$**: $(\lor)$ of $l$ distinct terms $t_1 \lor t_2 \lor \cdots t_l$.
- **Truth assignment** for $X$: is a function $\nu : X \to \{0, 1\}$.
- An assignment $\nu$ **satisfies** a clause $C$ if it causes at least one term in $C$ to evaluate to 1 (since $C$ is an $\lor$ of terms).
- An assignment **satisfies** a collection of clauses $C_1, C_2, \ldots C_k$ if it causes all clauses to evaluate to 1, i.e., $C_1 \land C_2 \land \cdots C_k = 1$.
  - $\nu$ 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**.
Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses:
  - $x_1 \lor \overline{x_2} \lor \overline{x_3}$
  - $x_2 \lor \overline{x_3} \lor x_4$
  - $x_3 \lor \overline{x_4}$

▶ Not a satisfying assignment

Assignment:
- $x_1 = 1$
- $x_2 = 0$
- $x_3 = 1$
- $x_4 = 0$

$x_1 \lor x_2 \lor x_3$
$x_2 \lor x_3 \lor x_4$
$x_3 \lor x_4$

▶ Is a satisfying assignment
Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses:
  \begin{align*}
  &x_1 \lor \overline{x_2} \lor \overline{x_3} \\
  &x_2 \lor \overline{x_3} \lor x_4 \\
  &x_3 \lor \overline{x_4}
  \end{align*}
- Assignment: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$
Example

- \( X = \{x_1, x_2, x_3, x_4\} \)
- Terms: \( x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4} \)
- Clauses:
  - \( x_1 \lor \overline{x_2} \lor \overline{x_3} \)
  - \( x_2 \lor \overline{x_3} \lor x_4 \)
  - \( x_3 \lor \overline{x_4} \)

- Assignment: \( x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0 \)
  - \( x_1 \lor \overline{x_2} \lor \overline{x_3} \)
  - \( x_2 \lor \overline{x_3} \lor x_4 \)
  - \( x_3 \lor \overline{x_4} \)
  - Not a satisfying assignment
Example

- \( X = \{x_1, x_2, x_3, x_4\} \)
- Terms: \( x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4} \)
- Clauses:
  
  \[
  \begin{align*}
  x_1 \lor \overline{x_2} \lor \overline{x_3} \\
  x_2 \lor \overline{x_3} \lor x_4 \\
  x_3 \lor \overline{x_4}
  \end{align*}
  \]
- Assignment: \( x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0 \)
  
  \[
  \begin{align*}
  x_1 \lor \overline{x_2} \lor \overline{x_3} \\
  x_2 \lor \overline{x_3} \lor x_4 \\
  x_3 \lor \overline{x_4}
  \end{align*}
  \]
  
  ▶ Not a satisfying assignment
- Assignment: \( x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0 \)
Example

- \( X = \{x_1, x_2, x_3, x_4\} \)
- Terms: \( x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4} \)
- Clauses:
  - \( x_1 \lor \overline{x_2} \lor \overline{x_3} \)
  - \( x_2 \lor \overline{x_3} \lor x_4 \)
  - \( x_3 \lor \overline{x_4} \)
- Assignment: \( x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0 \)
  - \( x_1 \lor \overline{x_2} \lor \overline{x_3} \)
  - \( x_2 \lor \overline{x_3} \lor x_4 \)
  - \( x_3 \lor \overline{x_4} \)
  - Not a satisfying assignment
- Assignment: \( x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0 \)
  - \( x_1 \lor \overline{x_2} \lor \overline{x_3} \)
  - \( x_2 \lor \overline{x_3} \lor x_4 \)
  - \( x_3 \lor \overline{x_4} \)
  - Is a satisfying assignment
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.
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

Is $C_1 \land C_2$ satisfiable?
Yes, by $x_1 = 1$, $x_2 = 1$.

Is $C_1 \land C_3$ satisfiable?
Yes, by $x_1 = 1$, $x_2 = 0$.

Is $C_2 \land C_3$ satisfiable?
Yes, by $x_1 = 0$, $x_2 = 1$.

Is $C_1 \land C_2 \land C_3$ satisfiable?
No.
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

1. Is $C_1 \land C_2$ satisfiable?
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

1. Is $C_1 \land C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$. 
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

1. Is $C_1 \land C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \land C_3$ satisfiable?
Examples of 3-SAT

Example:

- \( C_1 = x_1 \lor 0 \lor 0 \)
- \( C_2 = x_2 \lor 0 \lor 0 \)
- \( C_3 = \overline{x_1} \lor \overline{x_2} \lor 0 \)

1. Is \( C_1 \land C_2 \) satisfiable? Yes, by \( x_1 = 1, x_2 = 1 \).
2. Is \( C_1 \land C_3 \) satisfiable? Yes, by \( x_1 = 1, x_2 = 0 \).
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = x_1 \lor \overline{x}_2 \lor 0$

1. Is $C_1 \land C_2$ satisfiable? Yes, by $x_1 = 1$, $x_2 = 1$.
2. Is $C_1 \land C_3$ satisfiable? Yes, by $x_1 = 1$, $x_2 = 0$.
3. Is $C_2 \land C_3$ satisfiable?
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

1. Is $C_1 \land C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \land C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.
3. Is $C_2 \land C_3$ satisfiable? Yes, by $x_1 = 0, x_2 = 1$. 
Examples of 3-SAT

Example:

\[ C_1 = x_1 \lor 0 \lor 0 \]
\[ C_2 = x_2 \lor 0 \lor 0 \]
\[ C_3 = \overline{x_1} \lor \overline{x_2} \lor 0 \]

1. Is \( C_1 \land C_2 \) satisfiable? Yes, by \( x_1 = 1, x_2 = 1 \).
2. Is \( C_1 \land C_3 \) satisfiable? Yes, by \( x_1 = 1, x_2 = 0 \).
3. Is \( C_2 \land C_3 \) satisfiable? Yes, by \( x_1 = 0, x_2 = 1 \).
4. Is \( C_1 \land C_2 \land C_3 \) satisfiable?
Examples of 3-SAT

Example:

- $C_1 = x_1 \lor 0 \lor 0$
- $C_2 = x_2 \lor 0 \lor 0$
- $C_3 = \overline{x_1} \lor \overline{x_2} \lor 0$

1. Is $C_1 \land C_2$ satisfiable? Yes, by $x_1 = 1$, $x_2 = 1$.
2. Is $C_1 \land C_3$ satisfiable? Yes, by $x_1 = 1$, $x_2 = 0$.
3. Is $C_2 \land C_3$ satisfiable? Yes, by $x_1 = 0$, $x_2 = 1$.
4. Is $C_1 \land C_2 \land C_3$ satisfiable? No.
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

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

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

$C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$  
Select $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1.$

$C_2 = \overline{x_1} \lor x_2 \lor x_4$  
Choose one literal from each clause to evaluate to true.

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

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

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.
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 $C_i$. 
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.

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

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

- 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?

---

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

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

$C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4}$
Proving $3\text{-SAT} \leq_P \text{Independent Set}$

- $C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$
- $C_2 = \overline{x_1} \lor x_2 \lor x_4$
- $C_3 = \overline{x_1} \lor x_3 \lor \overline{x_4}$

Claim: $3\text{-SAT}$ instance is satisfiable iff $G$ has an independent set of size $k$.

Satisfiable assignment $\implies$ independent set of size $k$: 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 $= k$. Why?
Proving \(3\text{-SAT} \leq_P \text{Independent Set}\)

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

\[x_1 \quad v_{11} \quad v_{12} \quad v_{13}\]
\[\overline{x_2} \quad \overline{x_3}\]
\[v_{21} \quad x_1 \quad v_{22} \quad v_{23}\]
\[\overline{x_1}\]
\[v_{31} \quad \overline{x_1} \quad v_{32} \quad v_{33}\]
\[x_3 \quad x_4\]

- **Claim:** 3-SAT instance is satisfiable iff \(G\) has an independent set of size \(k\).
- **Satisfiable assignment \(\rightarrow\) independent set of size \(k\):** 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 \(= k\). Why?
- **Independent set \(S\) of size \(k\) \(\rightarrow\) satisfiable assignment:
Proving $3$-SAT $\leq_P$ Independent Set

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

Satisfiable assignment $\rightarrow$ independent set of size $k$: 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 $= k$. Why?

Independent set $S$ of size $k$ $\rightarrow$ satisfiable assignment: the size of this set is $k$. How do we construct a satisfying truth assignment from the nodes in the independent set?
Proving $3$-SAT $\leq_P$ Independent Set

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

Satisfiable assignment $\rightarrow$ independent set of size $k$: 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 $= k$. Why?

Independent set $S$ of size $k$ $\rightarrow$ 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 $k$.

- **Satisfiable assignment $\rightarrow$ independent set of size $k$:** 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 $= k$. Why?

- **Independent set $S$ of size $k$ $\rightarrow$ 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?

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

Diagram:

- $C_1 = x_1 \lor \overline{x_2} \lor \overline{x_3}$
- $C_2 = \overline{x_1} \lor x_2 \lor x_4$
- $C_3 = \overline{x_1} \lor x_3 \lor x_4$
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
  
  $3$-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ 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.
Introduction
Reductions
\( \mathcal{NP} \)
\( \mathcal{NP} \)-Complete
\( \mathcal{NP} \) vs. co-\( \mathcal{NP} \)

Problems and Algorithms

Primes

**INSTANCE:** A natural number \( n \)

**QUESTION:** Is \( n \) prime?

- Decision problem \( X \): for every input \( s \), answer \( X(s) \) is yes or no.
Primes

**INSTANCE:** A natural number $n$
**QUESTION:** Is $n$ prime?

- Decision problem $X$: for every input $s$, answer $X(s)$ is yes or no.
- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes, no}\}$.
- An algorithm $A$ *solves* the problem $X$ if for every input $s$,
  - if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
  - if $X(s) = \text{no}$ then $A(s) = \text{no}$
Primes

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

**QUESTION:** Is $n$ prime?

- Decision problem $X$: for every input $s$, answer $X(s)$ is yes or no.
- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes}, \text{no}\}$.
- An algorithm $A$ *solves* the problem $X$ if for every input $s$,
  - if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
  - if $X(s) = \text{no}$ then $A(s) = \text{no}$
- $A$ 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 $p(|s|) = |s|^{12}$ for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to $|s|^6$ by Pomerance and Lenstra, 2005).
Problems and Algorithms

PRIMES

INSTANCE: A natural number \( n \)

QUESTION: Is \( n \) prime?

- Decision problem \( X \): for every input \( s \), answer \( X(s) \) is yes or no.
- An algorithm \( A \) for a decision problem receives an input \( s \) and returns \( A(s) \in \{\text{yes, no}\} \).
- An algorithm \( A \) solves the problem \( X \) if for every input \( s \),
  - if \( X(s) = \text{yes} \) then \( A(s) = \text{yes} \) and
  - if \( X(s) = \text{no} \) then \( A(s) = \text{no} \)
- \( A \) 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 \( p(|s|) = |s|^{12} \) for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to \( |s|^6 \) by Pomerance and Lenstra, 2005).
- \( \mathcal{P} \): set of problems \( X \) for which there is a polynomial time algorithm.
Problems and Algorithms

**PRIMES**

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

**QUESTION:** Is $n$ prime?

- Decision problem $X$: for every input $s$, answer $X(s)$ is yes or no.
- An algorithm $A$ for a decision problem receives an input $s$ and returns $A(s) \in \{\text{yes, no}\}$.
- An algorithm $A$ solves the problem $X$ if for every input $s$,
  - if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
  - if $X(s) = \text{no}$ then $A(s) = \text{no}$
- $A$ 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 $p(|s|) = |s|^{12}$ for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to $|s|^{6}$ by Pomerance and Lenstra, 2005).
- $\mathcal{P}$: 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 that solves $X$. 

T. M. Murali
November 7, 12, 2018
NP and Computational Intractability
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 $X(s) = \text{yes}$.
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 \( X(s) = \text{yes} \).
- 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 \( s \)
     - \( X(s) = \text{yes} \) iff there is a certificate \( t \) such that \( B(s, t) = \text{yes} \) and
     - the size of \( t \) is polynomial in the size of \( s \).
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 $X(s) = \text{yes}$.
- 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 $s$
     - $X(s) = \text{yes}$ iff there is a certificate $t$ such that $B(s, t) = \text{yes}$ and
     - the size of $t$ is polynomial in the size of $s$.
- Certifier’s job is to take a candidate certificate ($t$) that $s \in X$ and check in polynomial time whether $t$ is a correct certificate.
- Certificate $t$ must be “short” so that certifier can run in polynomial time.
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 $X(s) = \text{yes}$.
- 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 $s$
     - $X(s) = \text{yes}$ iff there is a certificate $t$ such that $B(s, t) = \text{yes}$ and
     - the size of $t$ is polynomial in the size of $s$.
- Certifier’s job is to take a candidate certificate ($t$) that $s \in X$ and check in polynomial time whether $t$ is a correct certificate.
- Certificate $t$ must be “short” so that certifier can run in polynomial time.
- Certifier does not care about how to find these certificates.
\textbf{\textit{NP}}

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

- \( \mathcal{P} \): set of problems \( X \) for which there is a polynomial time algorithm.
- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \):

  - Certificate \( t \): a truth assignment to the variables.
  - Certifier \( B \): checks whether assignment causes each clause to evaluate to true.

  - 3-SAT \( \in \mathcal{NP} \):
    - Certificate \( t \): a set of at least \( k \) vertices.
    - Certifier \( B \): checks that no pair of these vertices are connected by an edge.

  - Set Cover \( \in \mathcal{NP} \):
    - Certificate \( t \): a list of \( k \) sets from the collection.
    - Certifier \( B \): checks if their union of these sets is \( U \).
$\mathcal{NP}$

- $\mathcal{P}$: set of problems $X$ for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- $3$-SAT $\in \mathcal{NP}$:
  - Certificate $t$: a truth assignment to the variables.
  - Certifier $B$:
\( \mathcal{NP} \)

- \( \mathcal{P} \): set of problems \( X \) for which there is a polynomial time algorithm.
- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- \( 3\text{-SAT} \in \mathcal{NP} \):
  - Certificate \( t \): a truth assignment to the variables.
  - Certifier \( B \): checks whether assignment causes each clause to evaluate to true.
\[ \mathcal{NP} \]

- \( \mathcal{P} \): set of problems \( X \) for which there is a polynomial time algorithm.
- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- 3-SAT \( \in \mathcal{NP} \):
  - Certificate \( t \): a truth assignment to the variables.
  - Certifier \( B \): checks whether assignment causes each clause to evaluate to true.
- Independent Set \( \in \mathcal{NP} \):
  - Certificate \( t \): a set of at least \( k \) vertices.
  - Certifier \( B \): checks that no pair of these vertices are connected by an edge.
$\mathcal{NP}$

- $\mathcal{P}$: set of problems $X$ for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$:
- Certificate $t$: a truth assignment to the variables.
- Certifier $B$: checks whether assignment causes each clause to evaluate to true.

Independent Set $\in \mathcal{NP}$:
- Certificate $t$: a set of at least $k$ vertices.
- Certifier $B$:
\[\mathcal{NP}\]

- \(\mathcal{P}\): set of problems \(X\) for which there is a polynomial time algorithm.
- \(\mathcal{NP}\) is the set of all problems for which there exists an efficient certifier.
- \(3\text{-SAT} \in \mathcal{NP}\):
  - Certificate \(t\): a truth assignment to the variables.
  - Certifier \(B\): checks whether assignment causes each clause to evaluate to true.
- \textbf{Independent Set} \(\in \mathcal{NP}\):
  - Certificate \(t\): a set of at least \(k\) vertices.
  - Certifier \(B\): checks that no pair of these vertices are connected by an edge.
\( \mathcal{NP} \)

- \( \mathcal{P} \): set of problems \( X \) for which there is a polynomial time algorithm.
- \( \mathcal{NP} \) is the set of all problems for which there exists an efficient certifier.
- 3-SAT \( \in \mathcal{NP} \):
  - Certificate \( t \): a truth assignment to the variables.
  - Certifier \( B \): checks whether assignment causes each clause to evaluate to true.
- Independent Set \( \in \mathcal{NP} \):
  - Certificate \( t \): a set of at least \( k \) vertices.
  - Certifier \( B \): checks that no pair of these vertices are connected by an edge.
- Set Cover \( \in \mathcal{NP} \):
  - Certificate \( t \): 
  - Certifier \( B \): 

\section*{\textbf{NP}}

- \textbf{P}: set of problems \(X\) for which there is a polynomial time algorithm.
- \(\text{NP}\) is the set of all problems for which there exists an efficient certifier.
- \(3\text{-SAT} \in \text{NP}\):
  - Certificate \(t\): a truth assignment to the variables.
  - Certifier \(B\): checks whether assignment causes each clause to evaluate to true.
- \textbf{Independent Set} \(\in \text{NP}\):
  - Certificate \(t\): a set of at least \(k\) vertices.
  - Certifier \(B\): checks that no pair of these vertices are connected by an edge.
- \textbf{Set Cover} \(\in \text{NP}\):
  - Certificate \(t\): a list of \(k\) sets from the collection.
  - Certifier \(B\):
\[
\mathcal{NP}
\]

- \(\mathcal{P}\): set of problems \(X\) for which there is a polynomial time algorithm.
- \(\mathcal{NP}\) is the set of all problems for which there exists an efficient certifier.

- \(3\text{-SAT} \in \mathcal{NP}\):
  - Certificate \(t\): a truth assignment to the variables.
  - Certifier \(B\): checks whether assignment causes each clause to evaluate to true.

- **Independent Set** \(\text{Set} \in \mathcal{NP}\):
  - Certificate \(t\): a set of at least \(k\) vertices.
  - Certifier \(B\): checks that no pair of these vertices are connected by an edge.

- **Set Cover** \(\text{Cover} \in \mathcal{NP}\):
  - Certificate \(t\): a list of \(k\) sets from the collection.
  - Certifier \(B\): checks if their union of these sets is \(U\).
Claim: \( \mathcal{P} \subseteq \mathcal{NP} \).
Claim: $P \subseteq NP$.

- Let $X$ be any problem in $P$.
- There is a polynomial time algorithm $A$ that solves $X$. 

\[ \text{P vs. NP} \]
Claim: $P \subseteq NP$.

- Let $X$ be any problem in $P$.
- There is a polynomial time algorithm $A$ that solves $X$.
- $B$ ignores $t$ and simply returns $A(s)$. Why is $B$ an efficient certifier?
Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
- Let $X$ be any problem in $\mathcal{P}$.
- There is a polynomial time algorithm $A$ that solves $X$.
- $B$ ignores $t$ and simply returns $A(s)$. Why is $B$ an efficient certifier?

Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$?
Claim: $P \subseteq NP$.

- Let $X$ be any problem in $P$.
- There is a polynomial time algorithm $A$ that solves $X$.
- $B$ ignores $t$ and simply 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.
Claim: $\mathcal{P} \subseteq \mathcal{NP}$.

- Let $X$ be any problem in $\mathcal{P}$.
- There is a polynomial time algorithm $A$ that solves $X$.
- $B$ ignores $t$ and simply returns $A(s)$. Why is $B$ an efficient certifier?

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.
Summary

- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VertexCover, SetCover, IndependentSet are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover
Summary

- $P \subseteq NP$
- $3$-SAT, VertexCover, SetCover, IndependentSet are in $NP$.
- $3$-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover
- What is the structure of the problems in $NP$?
Summary

NP

\[ P \subseteq NP \]

3-SAT, VertexCover, SetCover, IndependentSet are in \( NP \).

3-SAT \( \leq_P \) Independent Set \( \leq_P \) Vertex Cover \( \leq_P \) Set Cover

What is the structure of the problems in \( NP \)?

1. Is there a sequence of problems \( X_1, X_2, X_3, \ldots \) in \( NP \), such that \( X_1 \leq_P X_2 \leq_P X_3 \leq_P \ldots \)?
Summary

NP

- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VertexCover, SetCover, IndependentSet are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover
- What is the structure of the problems in $\mathcal{NP}$?
  1. Is there a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, such that $X_1 \leq_P X_2 \leq_P X_3 \leq_P \ldots$?
  2. Are there two problems $X_1$ and $X_2$ in $\mathcal{NP}$ such that there is no problem $X \in \mathcal{NP}$ where $X_1 \leq_P X$ and $X_2 \leq_P X$?
What are the hardest problems in $\text{NP}$?

A problem $X$ is $\text{NP}$-Complete if

(i) $X \in \text{NP}$

(ii) for every problem $Y \in \text{NP}$, $Y \leq_{P} X$.

A problem $X$ is $\text{NP}$-Hard if

(i) for every problem $Y \in \text{NP}$, $Y \leq_{P} X$.

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

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.
**NP-Complete and NP-Hard Problems**

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

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

1. \( X \in \mathcal{NP} \) and
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.

Does even one \( \mathcal{NP} \)-Complete problem exist?! If it does, how can we prove that every problem in \( \mathcal{NP} \) reduces to this problem?
\textbf{NP-Complete and NP-Hard Problems}

- What are the hardest problems in \textsf{NP}?

A problem \(X\) is \textbf{\textsf{NP-Complete}} if

1. \(X \in \textsf{NP}\) and
2. for every problem \(Y \in \textsf{NP}\), \(Y \leq_P X\).

A problem \(X\) is \textbf{\textsf{NP-Hard}} if

1. for every problem \(Y \in \textsf{NP}\), \(Y \leq_P X\).

\begin{center}
\begin{tikzpicture}
% Diagram code here
\end{tikzpicture}
\end{center}

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

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

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

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

for every problem \( Y \in \mathcal{NP} \),

\( Y \leq_P X \).

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

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.
**NP-Complete and NP-Hard Problems**

- What are the hardest problems in NP?

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

1. $X \in \mathcal{NP}$ and
2. for every problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

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

1. 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 NP that cannot be solved in polynomial time, then no NP-Complete problem can be solved in polynomial time.

**Does even one NP-Complete problem exist?!** If it does, how can we prove that every problem in NP reduces to this problem?
Circuit Satisfiability

- **Cook-Levin Theorem**: \textsc{Circuit Satisfiability} is 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 \( \land, \lor, \text{ or } \neg \).
  3. a single node with no outgoing edges represents the output of \( K \).

![Diagram of a circuit](image)

*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 \( \mathcal{NP} \)-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 \( \land \), \( \lor \), or \( \lnot \).
  3. a single node with no outgoing edges represents the output of \( K \).

**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.

▶ Skip proof; read textbook or Chapter 2.6 of Garey and Johnson.
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete
Proving Circuit Satisfiability is \( \mathcal{NP} \)-Complete

- Take an arbitrary problem \( X \in \mathcal{NP} \) and show that \( X \leq_P \text{Circuit Satisfiability} \).
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{Circuit Satisfiability}$.

- Claim we will not prove: any algorithm that takes a fixed number $n$ of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in $n$, the size of the circuit is a polynomial in $n$. 
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_p \text{Circuit Satisfiability}$.

- Claim we will not prove: any algorithm that takes a fixed number $n$ of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in $n$, the size of the circuit is a polynomial in $n$.

- To show $X \leq_p \text{Circuit Satisfiability}$, given an input $s$ of length $n$, we want to determine whether $s \in X$ using a black box that solves Circuit Satisfiability.
Proving Circuit Satisfiability is \( \mathcal{NP} \)-Complete

- Take an arbitrary problem \( X \in \mathcal{NP} \) and show that \( X \leq_P \text{Circuit Satisfiability}. \)
- Claim we will not prove: any algorithm that takes a fixed number \( n \) of bits as input and produces a yes/no answer
  
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in \( n \), the size of the circuit is a polynomial in \( n \).

- To show \( X \leq_P \text{Circuit Satisfiability} \), given an input \( s \) of length \( n \), we want to determine whether \( s \in X \) using a black box that solves CIRCUIT SATISFIABILITY.

- What do we know about \( X \)?
Proving Circuit Satisfiability is \( \mathcal{NP} \)-Complete

- Take an arbitrary problem \( X \in \mathcal{NP} \) and show that \( X \leq_P \text{Circuit Satisfiability} \).
- Claim we will not prove: any algorithm that takes a fixed number \( n \) of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in \( n \), the size of the circuit is a polynomial in \( n \).
- To show \( X \leq_P \text{Circuit Satisfiability} \), given an input \( s \) of length \( n \), we want to determine whether \( s \in X \) using a black box that solves \( \text{Circuit Satisfiability} \).
- What do we know about \( X \)? It has an efficient certifier \( B(\cdot, \cdot) \).
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{Circuit Satisfiability}$.

- Claim we will not prove: any algorithm that takes a fixed number $n$ of bits as input and produces a yes/no answer
  
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in $n$, the size of the circuit is a polynomial in $n$.

- To show $X \leq_P \text{Circuit Satisfiability}$, given an input $s$ of length $n$, we want to determine whether $s \in X$ using a black box that solves Circuit Satisfiability.

- What do we know about $X$? It has an efficient certifier $B(\cdot, \cdot)$.

- To determine whether $s \in X$, we ask “Is there a certificate $t$ of length $p(n)$ such that $B(s, t) = \text{yes}$?”
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

To determine whether $s \in X$, we ask “Is there a certificate $t$ of length $p(|s|)$ such that $B(s, t) = \text{yes}$?”
Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- To determine whether $s \in X$, we ask “Is there a certificate $t$ of length $p(|s|)$ such that $B(s, t) = \text{yes}$?”
- View $B(\cdot, \cdot)$ as an algorithm on $n + p(n)$ bits.
- Convert $B$ to a polynomial-sized circuit $K$ with $n + p(n)$ sources.
  1. First $n$ sources are hard-coded with the bits of $s$.
  2. The remaining $p(n)$ sources labelled with variables representing the bits of $t$. 
Proving Circuit Satisfiability is \( \mathcal{NP} \)-Complete

- To determine whether \( s \in X \), we ask “Is there a certificate \( t \) of length \( p(|s|) \) such that \( B(s, t) = \text{yes} \)?”
- View \( B(\cdot, \cdot) \) as an algorithm on \( n + p(n) \) bits.
- Convert \( B \) to a polynomial-sized circuit \( K \) with \( n + p(n) \) sources.
  1. First \( n \) sources are hard-coded with the bits of \( s \).
  2. The remaining \( p(n) \) sources labelled with variables representing the bits of \( t \).
- \( s \in X \) iff there is an assignment of the input bits of \( K \) that makes \( K \) satisfiable.
Example of Transformation to Circuit Satisfiability

- Does a graph $G$ on $n$ nodes have a two-node independent set?

$s$ encodes the graph $G$ with $\binom{n}{2}$ bits.
$t$ encodes the independent set with $n$ bits.
Certifier needs to check if
1. at least two bits in $t$ are set to 1 and
2. no two bits in $t$ are set to 1 if they form the ends of an edge (the corresponding bit in $s$ is set to 1).
Example of Transformation to Circuit Satisfiability

- Does a graph $G$ on $n$ nodes have a two-node independent set?
- $s$ encodes the graph $G$ with $\binom{n}{2}$ bits.
- $t$ encodes the independent set with $n$ bits.
- Certifier needs to check if
  1. at least two bits in $t$ are set to 1 and
  2. no two bits in $t$ are set to 1 if they form the ends of an edge (the corresponding bit in $s$ is set to 1).
Example of Transformation to Circuit Satisfiability

Suppose $G$ contains three nodes $u$, $v$, and $w$ with $v$ connected to $u$ and $w$. 
Example of Transformation to Circuit Satisfiability

- Suppose $G$ contains three nodes $u$, $v$, and $w$ with $v$ connected to $u$ and $w$.

\[ \text{Figure 8.5} \quad \text{A circuit to verify whether a 3-node graph contains a 2-node independent set.} \]
Asymmetry of Certification

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

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

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

Definition of efficient certification and \( NP \) is fundamentally asymmetric:

- An input \( s \) is a “yes” instance iff there exists a short certificate \( t \) such that \( B(s, t) = \text{yes} \).
- An input \( s \) is a “no” instance iff for all short certificates \( t \), \( B(s, t) = \text{no} \). The definition of \( NP \) does not guarantee a short proof for “no” instances.
For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of inputs $s$ such that $s \in \overline{X}$ iff $s \notin X$. 

\[ \text{co-}NP \]
co-$\mathcal{NP}$

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

If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
co-$\mathcal{NP}$

- For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of inputs $s$ 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 $\overline{X} \in \mathcal{NP}$?

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

Claim: If $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ then $\mathcal{P} \neq \mathcal{NP}$.
For a decision problem $X$, its complementary problem $\overline{X}$ is the set of inputs $s$ 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-\(NP\)**

- For a decision problem \(X\), its *complementary problem* \(\overline{X}\) is the set of inputs \(s\) 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-NP* iff \(\overline{X}\) belongs to \(\mathcal{NP}\).

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

- For a decision problem $X$, its *complementary problem* $\overline{X}$ is the set of inputs $s$ 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 $\mathcal{NP}$.

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

Claim: If $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ 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 $NP \cap co-NP$

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

- Problems in $NP \cap co-NP$ have a good characterisation.
Good Characterisations: the Class $\mathcal{NP} \cap \text{co-NP}$

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

- Problems in $\mathcal{NP} \cap \text{co-NP}$ 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 $\mathcal{NP} \cap \mathsf{co-NP}$

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

- Problems in $\mathcal{NP} \cap \mathsf{co-NP}$ 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: $\mathcal{P} \subseteq \mathcal{NP} \cap \mathsf{co-NP}$. 
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 $\nu$.

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

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