Reductions of NP-Complete Problems

- Circuit-Sat
  - 3-Sat
    - Ind-Set
      - Clique
      - Vertex-Cover
        - Set-Cover
    - HC
    - 3D-Matching
      - Subset-Sum
        - Knapsack
    - TSP
  - 3-Coloring
Recall: Independent Set Problem

**Def.** An independent set of $G = (V, E)$ is a subset $I \subseteq V$ such that no two vertices in $I$ are adjacent in $G$.

**Independent Set (Ind-Set) Problem**

**Input:** $G = (V, E), k$

**Output:** whether there is an independent set of size $k$ in $G$
3-Sat $\leq_P$ Ind-Set

$$(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$$
3-Sat $\leq_P$ Ind-Set

$(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- A clause $\Rightarrow$ a group of 3 vertices, one for each literal
- An edge between every pair of vertices in same group
3-Sat $\leq_P$ Ind-Set

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- A clause $\Rightarrow$ a group of 3 vertices, one for each literal
- An edge between every pair of vertices in same group
- An edge between every pair of contradicting literals
3-Sat $\leq_P$ Ind-Set

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- A clause $\Rightarrow$ a group of 3 vertices, one for each literal
- An edge between every pair of vertices in same group
- An edge between every pair of contradicting literals
- Problem: whether there is an IS of size $k = \#\text{clauses}$
3-Sat $\leq_p$ Ind-Set

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- A clause $\Rightarrow$ a group of 3 vertices, one for each literal
- An edge between every pair of vertices in same group
- An edge between every pair of contradicting literals
- Problem: whether there is an IS of size $k = \#\text{clauses}$

3-Sat instance is yes-instance $\iff$ Ind-Set instance is yes-instance:
3-Sat $\leq_P$ Ind-Set

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- A clause $\Rightarrow$ a group of 3 vertices, one for each literal
- An edge between every pair of vertices in same group
- An edge between every pair of contradicting literals
- Problem: whether there is an IS of size $k = \#\text{clauses}$

3-Sat instance is yes-instance $\iff$ Ind-Set instance is yes-instance:
- satisfying assignment $\Rightarrow$ independent set of size $k$
- independent set of size $k$ $\Rightarrow$ satisfying assignment
Satisfying Assignment \(\Rightarrow\) IS of Size \(k\):

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)
\]
Satisfying Assignment $\Rightarrow$ IS of Size $k$

- $$(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$$

- For every clause, at least 1 literal is satisfied
Satisfying Assignment $\Rightarrow$ IS of Size $k$

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every clause, at least 1 literal is satisfied

- Pick the vertex correspondent the literal
Satisfying Assignment $\Rightarrow$ IS of Size $\kappa$

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every clause, at least 1 literal is satisfied
- Pick the vertex correspondent the literal
- So, 1 literal from each group
Satisfying Assignment $\implies$ IS of Size $k$

- $\left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( x_2 \lor x_3 \lor x_4 \right) \land \left( \neg x_1 \lor \neg x_3 \lor x_4 \right)$

- For every clause, at least 1 literal is satisfied
- Pick the vertex correspondent the literal
- So, 1 literal from each group
- No contradictions among the selected literals
Satisfying Assignment $\Rightarrow$ IS of Size $k$

- $$(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$$

- For every clause, at least 1 literal is satisfied
- Pick the vertex correspondent to the literal
- So, 1 literal from each group
- No contradictions among the selected literals
- An IS of size $k$
\( IS \) of size \( k \) \( \Rightarrow \) Satisfying Assignment

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)
\]

For every group, exactly one literal is selected in IS.
No contradictions among the selected literals.
If \( x_i \) is selected in IS, set \( x_i = 1 \).
If \( \neg x_i \) is selected in IS, set \( x_i = 0 \).
Otherwise, set \( x_i \) arbitrarily.
IS of Size $k \Rightarrow$ Satisfying Assignment

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every group, exactly one literal is selected in IS
IS of Size $k \Rightarrow$ Satisfying Assignment

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every group, exactly one literal is selected in IS

- No contradictions among the selected literals

![Diagram](image-url)
IS of Size $k \implies$ Satisfying Assignment

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every group, exactly one literal is selected in IS
- No contradictions among the selected literals
- If $x_i$ is selected in IS, set $x_i = 1$
IS of Size $k \Rightarrow$ Satisfying Assignment

- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)$

- For every group, exactly one literal is selected in IS
- No contradictions among the selected literals
- If $x_i$ is selected in IS, set $x_i = 1$
- If $\neg x_i$ is selected in IS, set $x_i = 0$
\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor x_4)\]

For every group, exactly one literal is selected in IS

No contradictions among the selected literals

If \(x_i\) is selected in IS, set \(x_i = 1\)

If \(\neg x_i\) is selected in IS, set \(x_i = 0\)

Otherwise, set \(x_i\) arbitrarily
Reductions of NP-Complete Problems
Def. A **clique** in an undirected graph $G = (V, E)$ is a subset $S \subseteq V$ such that $\forall u, v \in S$ we have $(u, v) \in E$.
**Def.** A **clique** in an undirected graph $G = (V, E)$ is a subset $S \subseteq V$ such that $\forall u, v \in S$ we have $(u, v) \in E$.
**Def.** A **clique** in an undirected graph $G = (V, E)$ is a subset $S \subseteq V$ such that $\forall u, v \in S$ we have $(u, v) \in E$

---

**Clique Problem**

**Input:** $G = (V, E)$ and integer $k > 0$,

**Output:** whether there exists a clique of size $k$ in $G$
Def. A clique in an undirected graph \( G = (V, E) \) is a subset \( S \subseteq V \) such that \( \forall u, v \in S \) we have \( (u, v) \in E \).

Clique Problem

**Input:** \( G = (V, E) \) and integer \( k > 0 \),

**Output:** whether there exists a clique of size \( k \) in \( G \)

What is the relationship between Clique and Ind-Set?
Clique $\equiv_P$ Ind-Set

**Def.** Given a graph $G = (V, E)$, define $\overline{G} = (V, \overline{E})$ be the graph such that $(u, v) \in \overline{E}$ if and only if $(u, v) \notin E$.

**Obs.** $S$ is an independent set in $G$ if and only if $S$ is a clique in $\overline{G}$. 
Reductions of NP-Complete Problems

- Circuit-Sat
- 3-Sat
  - Ind-Set → Clique → Vertex-Cover → Set-Cover
  - HC
  - 3D-Matching
  - 3-Coloring
  - TSP
  - Subset-Sum
  - Knapsack
Def. Given a graph $G = (V, E)$, a vertex cover of $G$ is a subset $S \subseteq V$ such that for every $(u, v) \in E$ then $u \in S$ or $v \in S$. 

Vertex-Cover Problem
Input: $G = (V, E)$ and integer $k$
Output: whether there is a vertex cover of $G$ of size at most $k$. 

Diagram of a graph with vertices connected by edges.
**Vertex-Cover**

**Def.** Given a graph $G = (V, E)$, a *vertex cover* of $G$ is a subset $S \subseteq V$ such that for every $(u, v) \in E$ then $u \in S$ or $v \in S$. 
**Vertex-Cover**

**Def.** Given a graph $G = (V, E)$, a **vertex cover** of $G$ is a subset $S \subseteq V$ such that for every $(u, v) \in E$ then $u \in S$ or $v \in S$.

**Vertex-Cover Problem**

**Input:** $G = (V, E)$ and integer $k$

**Output:** whether there is a vertex cover of $G$ of size at most $k$
Vertex-Cover $=^P$ Ind-Set

Q: What is the relationship between Vertex-Cover and Ind-Set?

A: $S$ is a vertex-cover of $G = (V,E)$ if and only if $V \cap S$ is an independent set of $G$. 

---

57/75
Vertex-Cover $\equiv_P$ Ind-Set

Q: What is the relationship between Vertex-Cover and Ind-Set?
Vertex-Cover $=^p$ Ind-Set

**Q:** What is the relationship between Vertex-Cover and Ind-Set?

**A:** $S$ is a vertex-cover of $G = (V, E)$ if and only if $V \setminus S$ is an independent set of $G$. 
Reductions of NP-Complete Problems

Diagram:
- Circuit-Sat
  - 3-Sat
    - 3-Coloring
    - Subset-Sum
      - TSP
      - HC
    - Ind-Set
      - Vertex-Cover
        - Clique
        - Set-Cover
      - HC
A Strategy of Polynomial Reduction

Recall the definition of polynomial time reductions:

**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$. 
Recall the definition of polynomial time reductions:

**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$.

- In general, algorithm for $Y$ can call the algorithm for $X$ many times.
Recall the definition of polynomial time reductions:

**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$.

- In general, algorithm for $Y$ can call the algorithm for $X$ many times.
- However, for most reductions, we call algorithm for $X$ only once.
A Strategy of Polynomial Reduction

Recall the definition of polynomial time reductions:

**Def.** Given a black box algorithm \( A \) that solves a problem \( X \), if any instance of a problem \( Y \) can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to \( A \), then we say \( Y \) is polynomial-time reducible to \( X \), denoted as \( Y \leq_P X \).

- In general, algorithm for \( Y \) can call the algorithm for \( X \) many times.
- However, for most reductions, we call algorithm for \( X \) only once.
- That is, for a given instance \( s_Y \) for \( Y \), we only construct one instance \( s_X \) for \( X \).
A Strategy of Polynomial Reduction

- Given an instance $s_Y$ of problem $Y$, show how to construct in polynomial time an instance $s_X$ of problem such that:
  - $s_Y$ is a yes-instance of $Y \Rightarrow s_X$ is a yes-instance of $X$
  - $s_X$ is a yes-instance of $X \Rightarrow s_Y$ is a yes-instance of $Y$
Outline

1. Some Hard Problems
2. P, NP and Co-NP
3. Polynomial Time Reductions and NP-Completeness
4. NP-Complete Problems
5. Dealing with NP-Hard Problems
6. Summary
Q: How far away are we from proving or disproving $P = NP$?
Q: How far away are we from proving or disproving $P = NP$?

Try to prove an “unconditional” lower bound on running time of algorithm solving a NP-complete problem.
Q: How far away are we from proving or disproving \( P = NP \)?

- Try to prove an “unconditional” lower bound on running time of algorithm solving a NP-complete problem.
- For 3-Sat problem:
Q: How far away are we from proving or disproving P = NP?

- Try to prove an “unconditional” lower bound on running time of algorithm solving a NP-complete problem.
- For 3-Sat problem:
  - Assume the number of clauses is $\Theta(n)$, $n =$ number variables
Q: How far away are we from proving or disproving P = NP?

- Try to prove an “unconditional” lower bound on running time of
  algorithm solving a NP-complete problem.
- For 3-Sat problem:
  - Assume the number of clauses is $\Theta(n)$, $n =$ number variables
  - Best algorithm runs in time $O(c^n)$ for some constant $c > 1$
Q: How far away are we from proving or disproving $P = NP$?

- Try to prove an “unconditional” lower bound on running time of algorithm solving a NP-complete problem.
- For 3-Sat problem:
  - Assume the number of clauses is $\Theta(n)$, $n =$ number variables
  - Best algorithm runs in time $O(c^n)$ for some constant $c > 1$
  - Best lower bound is $\Omega(n)$
Q: How far away are we from proving or disproving \( P = NP \)?

- Try to prove an “unconditional” lower bound on running time of algorithm solving a NP-complete problem.
- For 3-Sat problem:
  - Assume the number of clauses is \( \Theta(n) \), \( n = \) number variables
  - Best algorithm runs in time \( O(c^n) \) for some constant \( c > 1 \)
  - Best lower bound is \( \Omega(n) \)
- Essentially we have no techniques for proving lower bound for running time
Dealing with NP-Hard Problems

- Faster exponential time algorithms
- Solving the problem for special cases
- Fixed parameter tractability
- Approximation algorithms
Faster Exponential Time Algorithms

3-SAT:

Brute-force: $O(2^n \cdot \text{poly}(n))$

Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:

Brute-force: $O(n! \cdot \text{poly}(n))$

Better algorithm: $O(2^n \cdot \text{poly}(n))$

In practice: TSP Solver can solve Euclidean TSP instances with more than 100,000 vertices
Faster Exponential Time Algorithms

3-SAT:

- Brute-force: $O(2^n \cdot \text{poly}(n))$
Faster Exponential Time Algorithms

3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$

Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:
- Brute-force: $O(n! \cdot \text{poly}(n))$
- Better algorithm: $O(2^n \cdot \text{poly}(n))$
- In practice: TSP Solver can solve Euclidean TSP instances with more than 100,000 vertices
Faster Exponential Time Algorithms

3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$
- Practical SAT Solver: solves real-world sat instances with more than 10,000 variables
Faster Exponential Time Algorithms

3-SAT:
- Brute-force: $O\left(2^n \cdot \text{poly}(n)\right)$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$
- Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:
Faster Exponential Time Algorithms

3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$
- Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:
- Brute-force: $O(n! \cdot \text{poly}(n))$
3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$
- Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:
- Brute-force: $O(n! \cdot \text{poly}(n))$
- Better algorithm: $O(2^n \cdot \text{poly}(n))$
3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.844^n \rightarrow 1.3334^n$
- Practical SAT Solver: solves real-world sat instances with more than 10,000 variables

Travelling Salesman Problem:
- Brute-force: $O(n! \cdot \text{poly}(n))$
- Better algorithm: $O(2^n \cdot \text{poly}(n))$
- In practice: TSP Solver can solve Euclidean TSP instances with more than 100,000 vertices
Maximum independent set problem is NP-hard on general graphs, but easy on trees, bounded tree-width graphs, interval graphs, ···
Solving the problem for special cases

Maximum independent set problem is NP-hard on general graphs, but easy on

- trees
Solving the problem for special cases

Maximum independent set problem is NP-hard on general graphs, but easy on

- trees
- bounded tree-width graphs
Solving the problem for special cases

Maximum independent set problem is NP-hard on general graphs, but easy on
- trees
- bounded tree-width graphs
- interval graphs
Solving the problem for special cases

Maximum independent set problem is NP-hard on general graphs, but easy on
- trees
- bounded tree-width graphs
- interval graphs
- ...
Solving the problem for special cases

Collaborative delivery problem (reduction from 3DM) is NP-hard on general graphs, but easy on...
Collaborative delivery problem (reduction from 3DM) is NP-hard on general graphs, but easy on path (HW2 Problem 2)
Solving the problem for special cases

Collaborative delivery problem (reduction from 3DM) is NP-hard on general graphs, but easy on
- path (HW2 Problem 2)
- trees
Solving the problem for special cases

Collaborative delivery problem (reduction from 3DM) is NP-hard on general graphs, but easy on
- path (HW2 Problem 2)
- trees
- …
Problem: whether there is a vertex cover of size $k$, for a small $k$ (number of nodes is $n$, number of edges is $\Theta(n)$.)
Fixed Parameter Tractability

- Problem: whether there is a vertex cover of size $k$, for a small $k$ (number of nodes is $n$, number of edges is $\Theta(n)$.)
- Brute-force algorithm: $O(kn^{k+1})$
Problem: whether there is a vertex cover of size \( k \), for a small \( k \) (number of nodes is \( n \), number of edges is \( \Theta(n) \)).

Brute-force algorithm: \( O(kn^{k+1}) \)

Better running time: \( O(2^k \cdot kn) \)
Problem: whether there is a vertex cover of size $k$, for a small $k$ (number of nodes is $n$, number of edges is $\Theta(n)$.)

- Brute-force algorithm: $O(kn^{k+1})$
- Better running time: $O(2^k \cdot kn)$
- Running time is $f(k)n^c$ for some $c$ independent of $k$
Problem: whether there is a vertex cover of size \( k \), for a **small** \( k \) (number of nodes is \( n \), number of edges is \( \Theta(n) \)).

- Brute-force algorithm: \( O(kn^{k+1}) \)
- Better running time: \( O(2^k \cdot kn) \)
- Running time is \( f(k)n^c \) for some \( c \) independent of \( k \)
- Vertex-Cover is fixed-parameter tractable.
Approximation Algorithms

- For optimization problems, approximation algorithms will find sub-optimal solutions in \textit{polynomial time}.
Approximation Algorithms

- For optimization problems, approximation algorithms will find sub-optimal solutions in polynomial time.
- **Approximation ratio** is the ratio between the quality of the solution output by the algorithm and the quality of the optimal solution.

There is a 2-approximation for the vertex cover problem: we can efficiently find a vertex cover whose size is at most 2 times that of the optimal vertex cover.
Approximation Algorithms

- For optimization problems, approximation algorithms will find sub-optimal solutions in polynomial time.
- **Approximation ratio** is the ratio between the quality of the solution output by the algorithm and the quality of the optimal solution.
- We want to make the approximation ratio as small as possible, while maintaining the property that the algorithm runs in polynomial time.

There is a 2-approximation for the vertex cover problem: we can efficiently find a vertex cover whose size is at most 2 times that of the optimal vertex cover.
For optimization problems, approximation algorithms will find sub-optimal solutions in polynomial time.

**Approximation ratio** is the ratio between the quality of the solution output by the algorithm and the quality of the optimal solution. We want to make the approximation ratio as small as possible, while maintaining the property that the algorithm runs in polynomial time.

There is an 2-approximation for the vertex cover problem: we can efficiently find a vertex cover whose size is at most 2 times that of the optimal vertex cover.
2-Approximation Algorithm for Vertex Cover

Let the set $C$ and $C^*$ be the sets output by above algorithm and an optimal alg, respectively. Let $S$ be the set of edges selected.

Since no two edge in $S$ are covered by the same vertex (Once an edge is picked in line 3, all other edges that are incident on its endpoints are removed from $E$ in line 4), we have $|C^*| \geq |S|$;

As we have added both vertices of edge $(u, v)$, we get $|C| = 2|S|$ but $C^*$ have to add one of the two, thus, $|C|/|C^*| \leq 2$. 