Unit 9: Coping with NP-Completeness

• Course contents:
  - Complexity classes
  - Reducibility and NP-completeness proofs
  - Coping with NP-complete problems

• Reading:
  - Chapter 34
  - Chapter 35.1
Complexity Classes

- **The class P**: class of problems that can be solved in polynomial time in the **size of input**.
  - *Size of input*: size of encoded “binary” strings.
  - Edmonds: Problems in P are considered **tractable**.
  - Closed under addition, multiplication, composition, complement, etc.
- **The class NP (Nondeterministic Polynomial)**: class of problems that can be verified in polynomial time in the size of input.
  - P = NP?
- **The class NP-complete (NPC)**: Any NPC problem can be solved in polynomial time ⇒ **All** problems in NP can be solved in polynomial time.

![Diagram showing the relationship between P, NPC, and NP under the assumption P ≠ NP or P = NP]

- most likely case: P
- if P ≠ NP
- NPC
- if P = NP
- P = NP = NPC
Verification Algorithm and Class NP

• **Verification algorithm**: a 2-argument algorithm $A$, where one argument is an input string $x$ and the other is a binary string $y$ (called a **certificate**). $A$ verifies $x$ if there exists $y$ s.t. $A$ answers “yes.”

• **Exp**: The **Traveling Salesman Problem (TSP)**
  - **Instance**: a set of $n$ cities, distance between each pair of cities, and a bound $B$.
  - **Question**: is there a route that starts and ends at a given city, visits every city exactly once, and has total distance $\leq B$?

• Is TSP $\in$ NP?
  - Need to **check** a solution in polynomial time.
    - Guess a tour (certificate).
    - Check if the tour visits every city exactly once.
    - Check if the tour returns to the start.
    - Check if total distance $\leq B$.

• All can be done in $O(n)$ time, so TSP $\in$ NP.
Complexity Classes NP and co-NP

• Is class NP closed under complement?
• **Class co-NP**: class of problems whose *complement* problems are in NP.
  - \( \text{co-NP} = \{ L : \overline{L} \in \text{NP} \} \).

**TSP-Complement:**
  - **Instance:** a set of \( n \) cities, distance between each pair of cities, and a bound \( B \).
  - **Question:** are all tours that start and end at a given city, visit every city exactly once, and have total distance > \( B \)?

**TSP-Complement \( \in \) NP?**
  - Equivalently, TSP \( \in \) co-NP?

---

![Diagram of relationships among P, NP, co-NP](attachment:diagram.png)

four possibilities for relationships among P, NP, co-NP

(d) most likely case
Decision & Optimization Problems

• **Decision problems**: those having yes/no answers.
  - MST: Given a graph \( G=(V, E) \) and a bound \( K \), is there a spanning tree with a cost at most \( K \)?
  - TSP: Given a set of cities, distance between each pair of cities, and a bound \( B \), is there a route that starts and ends at a given city, visits every city exactly once, and has total distance at most \( B \)?

• **Optimization problems**: those finding a legal configuration such that its cost is minimum (or maximum).
  - MST: Given a graph \( G=(V, E) \), find the cost of a minimum spanning tree of \( G \).
  - TSP: Given a set of cities and that distance between each pair of cities, find the distance of a “minimum route” starts and ends at a given city and visits every city exactly once.
Decision v.s. Optimization Problems

- Could apply binary search on a decision problem to obtain solutions to its optimization problem.
- NP-completeness is associated with decision problems.
- c.f., Optimal solutions/costs, optimal (exact) algorithms
  - optimal ≠ exact in the theoretic computer science community.
Polynomial-time Reduction

- **Motivation:** Let $L_1$ and $L_2$ be two decision problems. Suppose algorithm $A_2$ can solve $L_2$. Can we use $A_2$ to solve $L_1$?

- **Polynomial-time reduction $f$ from $L_1$ to $L_2$: $L_1 \leq_P L_2$**
  - $f$ reduces input for $L_1$ into an input for $L_2$ s.t. the reduced input is a “yes” input for $L_2$ iff the original input is a “yes” input for $L_1$.
  - $L_1 \leq_P L_2$ if $\exists$ polynomial-time computable function $f$: $\{0, 1\}^* \to \{0, 1\}^*$ s.t. $x \in L_1$ iff $f(x) \in L_2$, $\forall x \in \{0, 1\}^*$.
  - $L_2$ is at least as hard as $L_1$.

- $f$ is computable in polynomial time.
Significance of Reduction

- Significance of $L_1 \leq_P L_2$:
  - $\exists$ polynomial-time algorithm for $L_2 \Rightarrow \exists$ polynomial-time algorithm for $L_1$ ($L_2 \in P \Rightarrow L_1 \in P$).
  - $\forall$ polynomial-time algorithm for $L_1 \Rightarrow \forall$ polynomial-time algorithm for $L_2$ ($L_1 \not\in P \Rightarrow L_2 \not\in P$).
- $\leq_P$ is transitive, i.e., $L_1 \leq_P L_2$ and $L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$.
System of Difference Constraints Revisited

- Example reduction from the system of difference constraints problem to the shortest path one.

- **Constraint graph**: Weighted, directed graph $G=(V, E)$.
  - $V = \{ v_0, v_1, \ldots, v_n \}$
  - $E = \{(v_i, v_j): x_j - x_i \leq b_k\} \cup \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_n)\}$
  - $w(v_i, v_j) = b_k$ if $x_j - x_i \leq b_k$ is a difference constraint.

- If $G$ contains no negative-weight cycle, then $x = (\delta(v_0, v_1), \delta(v_0, v_2), \ldots, \delta(v_0, v_n))$ is a feasible solution; no feasible solution, otherwise.

\[
\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)
\]

\[
\begin{align*}
x_1 - x_2 & \leq 0 \\
x_1 - x_5 & \leq -1 \\
x_2 - x_5 & \leq 1 \\
x_3 - x_1 & \leq 5 \\
x_4 - x_1 & \leq 4 \\
x_4 - x_3 & \leq -1 \\
x_5 - x_3 & \leq -3 \\
x_5 - x_4 & \leq -3
\end{align*}
\]
Maximum Cardinality Bipartite Matching Revisited

- Example reduction from the matching problem to the max-flow one.
- Given a bipartite graph $G = (V, E)$, $V = L \cup R$, construct a unit-capacity flow network $G' = (V', E')$:
  
  $V' = V \cup \{s, t\}$
  
  $E' = \{(s, u): u \in L\} \cup \{(u, v): u \in L, v \in R, (u, v) \in E\} \cup \{(v, t): v \in R\}$.
- The cardinality of a maximum matching in $G = |M|$ is the value of a maximum flow in $G'$ (i.e., $|M| = |f|$). 

![Diagram of bipartite graph and flow network]

$|M| = 3$

$|f| = 3$
Polynomial Reduction: HC \( \leq_p \) TSP

- **The Hamiltonian Circuit Problem (HC)**
  - **Instance:** an undirected graph \( G = (V, E) \).
  - **Question:** is there a cycle in \( G \) that includes every vertex exactly once?

- **TSP: The Traveling Salesman Problem**

- **Claim:** HC \( \leq_p \) TSP.
  1. Define a function \( f \) mapping any HC instance into a TSP instance, and show that \( f \) can be computed in polynomial time.
  2. Prove that \( G \) has an HC iff the reduced instance has a TSP tour with distance \( \leq B \) (\( x \in \text{HC} \iff f(x) \in \text{TSP} \)).
1. Define a reduction function $f$ for $HC \leq_P TSP$.

   - Given an HC instance $G = (V, E)$ with $n$ vertices
     
     - Create a set of $n$ cities labeled with names in $V$.
     
     - Assign distance between $u$ and $v$
       
       $d(u, v) = \begin{cases} 
       1, & \text{if } (u, v) \in E, \\
       2, & \text{if } (u, v) \notin E. 
       \end{cases}$

     - Set bound $B = n$.

   - $f$ can be computed in $O(V^2)$ time.
**HC \leq_p TSP: Step 2**

2. \( G \) has a HC iff the reduced instance has a TSP with distance \( \leq B \).

- \( x \in HC \Rightarrow f(x) \in TSP \).
  - Suppose the HC is \( h = \langle v_1, v_2, \ldots, v_n, v_1 \rangle \). Then, \( h \) is also a tour in the transformed TSP instance.
  - The distance of the tour \( h \) is \( n = B \) since there are \( n \) consecutive edges in \( E \), and so has distance 1 in \( f(x) \).
  - Thus, \( f(x) \in TSP \) (\( f(x) \) has a TSP tour with distance \( \leq B \)).
2. G has a HC iff the reduced instance has a TSP with distance \( \leq B \).
   
   - \( f(x) \in \text{TSP} \Rightarrow x \in \text{HC} \).
   - Suppose there is a TSP tour with distance \( \leq n = B \). Let it be \( <v_1, v_2, \ldots, v_n, v_1> \).
   - Since distance of the tour \( \leq n \) and there are \( n \) edges in the TSP tour, the tour contains only edges in \( E \).
   - Thus, \( <v_1, v_2, \ldots, v_n, v_1> \) is a Hamiltonian cycle (\( x \in \text{HC} \)).
**HP Application: Crosstalk Minimization**

- **Goal:** Find a wire ordering s.t. the overall crosstalk is minimized.
  - Construct a complete weighted graph $G = (V, E, W)$: $V \leftrightarrow$ wires,
  - $W \leftrightarrow$ coupling length between each pair of wires.
  - The minimum weighted Hamiltonian path induces the minimum crosstalk.

![Diagram showing wire ordering and crosstalk minimization](image)
NP-Completeness

• A **decision** problem $L$ (a language $L \subseteq \{0, 1\}^*$) is **NP-complete (NPC)** if
  1. $L \in \text{NP}$, and
  2. $L' \leq_P L$ for every $L' \in \text{NP}$.

• **NP-hard**: If $L$ satisfies property 2, but not necessarily property 1, we say that $L$ is **NP-hard**.

• Suppose $L \in \text{NPC}$.
  - If $L \in P$, then there exists a polynomial-time algorithm for every $L' \in \text{NP}$ (i.e., $P = \text{NP}$).
  - If $L \notin P$, then there exists no polynomial-time algorithm for any $L' \in \text{NPC}$ (i.e., $P \neq \text{NP}$).
Proving NP-Completeness

- Five steps for proving that $L$ is NP-complete:
  1. Prove $L \in \text{NP}$.  
  2. Select a known NP-complete problem $L'$.  
  3. Construct a reduction $f$ transforming every instance of $L'$ to an instance of $L$.  
  4. Prove that $x \in L'$ iff $f(x) \in L$ for all $x \in \{0, 1\}^*$.  
  5. Prove that $f$ is a polynomial-time transformation.
The Circuit-Satisfiability Problem (Circuit-SAT)

- **The Circuit-Satisfiability Problem (Circuit-SAT):**
  - **Instance:** A combinational circuit $C$ composed of AND, OR, and NOT gates.
  - **Question:** Is there an assignment of Boolean values to the inputs that makes the output of $C$ to be 1?

- A circuit is satisfiable if there exists a set of Boolean input values that makes the output of the circuit to be 1.
  - Circuit (a) is satisfiable since $<x_1, x_2, x_3> = <1, 1, 0>$ makes the output to be 1.

- **Circuit-SAT is NP-complete.** (Cook, ACM STOC'71)
  - Circuit-SAT $\in$ NP.
  - $\forall L' \in$ NP, $L' \leq_p$ Circuit-SAT.
Structure of NP-Completeness Proofs

Circuit–SAT
  ↓
  local replacement
SAT
  ↓
  local replacement
3SAT
  ↓
  component design
Clique
  ↓
  local replacement
  ↓
Independent–Set
  ↓
  restriction
Vertex–Cover
  ↓
  component design (omitted)
Hitting–Set
HC
  ↓
TSP
  ↓
  component design (omitted)
  ↓
Subset–Sum
The Satisfiability Problem (SAT)

• The Satisfiability Problem (SAT):
  – **Instance:** A Boolean formula $\phi$.
  – **Question:** Is there an assignment of truth values to the variables that makes $\phi$ true?

• **Truth assignment:** set of values for the variables of $\phi$.

• **Satisfying assignment:** a truth assignment that makes $\phi$ evaluate to 1.

• **Exp:** $\phi = ((x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2$
  – Truth assignment: $<x_1, x_2, x_3, x_4> = <0, 0, 1, 1>, <0, 1, 0, 1>, \text{etc.}$
  – Satisfying assignment: $<x_1, x_2, x_3, x_4> = <0, 0, 1, 1>, \text{etc.}$

• **Satisfiable formula:** a formula with a satisfying assignment.
  – $\phi$ is a satisfiable formula.
SAT is NP-Complete

1. SAT ∈ NP.

2. SAT is NP-hard: Prove that Circuit-SAT ≤ \text{P} SAT.
   - For each wire \( x_i \) in circuit \( C \), the formula \( \phi \) has a variable \( x_i \).
   - The operation of a gate is expressed as a formula with associated variables, e.g., \( x_{10} \leftrightarrow (x_7 \land x_8 \land x_9) \).
   - \( \phi = \text{AND} \) of the circuit-output variable with the conjunction (\( \land \)) of clauses describing the operation of each gate, e.g.,
     \[
     \phi = x_{10} \land (x_4 \leftrightarrow \neg x_3) \land (x_5 \leftrightarrow (x_1 \lor x_2)) \land (x_6 \leftrightarrow \neg x_4) \\
     \land (x_7 \leftrightarrow (x_1 \land x_2 \land x_4)) \land (x_8 \leftrightarrow (x_5 \lor x_6)) \\
     \land (x_9 \leftrightarrow (x_6 \lor x_7)) \land (x_{10} \leftrightarrow (x_7 \land x_8 \land x_9))
     \]
   - Circuit \( C \) is satisfiable \( \Leftrightarrow \) formula \( \phi \) is satisfiable. (Why?)
   - Given a circuit \( C \), it takes polynomial time to construct \( \phi \).
3SAT is NP-Complete

- **3SAT**: Satisfiability of boolean formulas in **3-conjunctive normal form (3-CNF)**.
  - Each clause has exactly 3 distinct literals, e.g., \( \phi = (x_1 \lor \neg x_1 \lor x_2) \land (x_3 \lor x_2 \lor x_4) \land (\neg x_2 \lor x_3 \lor \neg x_4) \)
- **3SAT \( \in \text{NP} \)** (will omit this part for other proofs).
- **3SAT is NP-hard**: SAT \( \leq_p \) 3SAT.
  1. Construct a binary “parse” tree for input formula \( \phi \) and introduce a variable \( y_i \) for the output of each internal node.
    \[ \phi = ((x_1 \rightarrow x_2) \lor \neg ((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2. \]
3SAT is NP-Complete (cont'd)

2. Rewrite $\phi$ as the AND of the root variable and a conjunction of clauses describing the operation of each node.

$$\phi' = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2)) \land (y_2 \leftrightarrow (y_3 \lor y_4)) \land (y_3 \leftrightarrow (x_1 \rightarrow x_2))$$
$$\land (y_4 \leftrightarrow \neg y_5) \land (y_5 \leftrightarrow (y_6 \lor x_4)) \land (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)).$$

3. Convert each clause $\phi'_i$ into CNF.

- Construct the **disjunctive normal form** for $\neg \phi'_i$ and then apply DeMorgan's law to get the CNF formula $\phi''_i$.
- E.g., $\neg \phi'_1 = (y_1 \land y_2 \land x_2) \lor (y_1 \land \neg y_2 \land x_2) \lor (y_1 \land \neg y_2 \land \neg x_2) \lor (\neg y_1 \land y_2 \land \neg x_2)$
- $\phi''_1 = \neg (\neg \phi'_1) = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor x_2) \land (y_1 \lor \neg y_2 \lor x_2)$.
4. Make each clause $C_i$ have **exactly** 3 distinct literals to get $\phi'''$.

- $C_i$ has 3 distinct literals: do nothing.
- $C_i$ has 2 distinct literals: $C_i = (l_1 \lor l_2) = (l_1 \lor l_2 \lor p) \land (l_1 \lor l_2 \lor \neg p)$.
- $C_i$ has only 1 literal: $C_i = l = (l \lor p \lor q) \land (l \lor \neg p \lor q) \land (l \lor p \lor \neg q) \land (l \lor \neg p \lor \neg q)$.

**Claim:** The 3-CNF formula $\phi'''$ is satisfiable $\iff \phi$ is satisfiable.

- All transformations can be done in polynomial time.

```
<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_2$</th>
<th>$(y_1 \iff (y_2 \land \neg x_2))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

truth table for $(y_1 \iff (y_2 \land \neg x_2))$
**Clique is NP-Complete**

- **A clique in** $G = (V, E)$ **is a complete subgraph of** $G$.
- **The Clique Problem (Clique)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - **Question:** is there a clique $V' \subseteq V$ of size $\geq k$?
- **Clique $\in$ NP.**
- **Clique is NP-hard:** $3\text{SAT} \leq_p \text{Clique}.$
  - **Key:** Construct a graph $G$ such that $\phi$ is satisfiable $\iff G$ has a clique of size $k$.

![Diagram of clique problem]
3SAT \leq_p \text{Clique}

- Let $\phi = C_1 \land C_2 \land \ldots \land C_k$ be a Boolean formula in 3-CNF with $k$ clauses. Each $C_r$ has exactly 3 distinct literals $l_1^r, l_2^r, l_3^r$.
- For each $C_r = (l_1^r \lor l_2^r \lor l_3^r)$ in $\phi$, introduce a triple of vertices $v_{r1}, v_{r2}, v_{r3}$ in $V$.
- Build an edge between $v_i^r, v_j^s$ if both of the following hold:
  - $v_i^r$ and $v_j^s$ are in different triples, and
  - $l_i^r$ is not the negation of $l_j^s$
- $G$ can be computed from $\phi$ in polynomial time.

Satisfying assignment: $\langle x_1, x_2, x_3 \rangle = \langle x, 0, 1 \rangle$
\( \phi \) is satisfiable \( \iff \) G has a clique of size \( k \)

- \( \phi \) is satisfiable \( \Rightarrow \) G has a clique of size \( k \).
  - \( \phi \) is satisfiable \( \Rightarrow \) each \( C_r \) contains at least one \( l_i^r = 1 \) and each such literal corresponds to a vertex \( v_i^r \).
  - Picking a “true” literal from each \( C_r \) forms a set of \( V' \) of \( k \) vertices.
  - For any two vertices \( v_i^r, v_j^s \in V', r \neq s, l_i^r = l_j^s = 1 \) and thus \( l_i^r, l_j^s \) cannot be complements. Thus, edge \((v_i^r, v_j^s) \in E\).
\( \phi \) Is Satisfiable \( \iff \) \( G \) Has a Clique of Size \( k \)

- \( G \) has a clique of size \( k \) \( \Rightarrow \) \( \phi \) is satisfiable.
  - \( G \) has a clique \( V' \) of size \( k \) \( \Rightarrow \) \( V' \) contains exactly one vertex per triple since no edges connect vertices in the same triple.
  - Assign 1 to each \( l_i' \) such that \( v_i' \in V' \) \( \Rightarrow \) each \( C_r \) is satisfied, and so is \( \phi \).

\[
C_1 = x_1 \lor \neg x_2 \lor \neg x_3
\]

\[
C_2 = \neg x_1 \lor x_2 \lor x_3
\]

\[
C_3 = x_1 \lor x_2 \lor x_3
\]

satisfying assignment: \( <x_1, x_2, x_3> = <x, 0, 1> \)
**Vertex-Cover is NP-Complete**

- A **vertex cover** of $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$.

- **The Vertex-Cover Problem (Vertex-Cover)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \leq |V|$.
  - **Question:** is there a subset $V' \subseteq V$ of size $\leq k$ such that each edge in $E$ has at least one vertex (endpoint) in $V'$?

- **Vertex-Cover $\in$ NP.**
- **Vertex-Cover is NP-hard:** Clique $\leq_P$ Vertex-Cover.
  - **Key:** complement of $G$: $\overline{G} = (V, \overline{E})$, $\overline{E} = \{(u, v): (u, v) \notin E\}$.

![Diagram of vertex cover and complement]

- Clique $V' = \{u, v, x, y\}$
- Vertex cover $V - V' = \{w, z\}$
**Vertex-Cover is NP-Complete (cont'd)**

- **G** Has a Clique of **Size k** $\Rightarrow$ **$\overline{G}$** Has a Vertex Cover of size $|V| - k$.
  - Suppose that **G** has a clique $V' \subseteq V$ with $|V'| = k$.
  - Let $(u, v)$ be any edge in $\overline{E}$ $\Rightarrow$ $(u, v) \not\in E$ $\Rightarrow$ at least one of $u$ or $v$ does not belong to $V'$.
  - So, $u \in V - V'$ or $v \in V - V'$ $\Rightarrow$ edge $(u, v)$ is covered by $V - V'$.
  - Thus, $V - V'$ forms a vertex cover of $\overline{G}$, and $|V - V'| = |V| - k$.

\[
\begin{align*}
\text{clique } V' &= \{u, v, x, y\} \\
\text{vertex cover } V - V' &= \{w, z\}
\end{align*}
\]
Vertex-Cover is NP-Complete (cont'd)

- Has a Vertex Cover of size $|V|-k \Rightarrow G$ Has a Clique of Size $k$.
  - Suppose that $\overline{G}$ has a vertex cover $V' \subseteq V$ with $|V'| = |V| - k$.
  - $\forall \ u, \ v \in V$, if $(u, v) \in \overline{E}$, then $u \in V'$ or $v \in V'$ or both.
  - So, $\forall \ u, \ v \in V$, if $u \notin V'$ and $v \notin V'$, $(u, v) \in E \Rightarrow V - V'$ is a clique, and $|V| - |V'| = k$.

clique $V' = \{u, v, x, y\}$

vertex cover $V - V' = \{w, z\}$
Clique, Independent-Set, Vertex-Cover

- **An independent set** of \( G = (V, E) \) is a subset \( V' \subseteq V \) such that \( G \) has no edge between any pair of vertices in \( V' \).

- **The Independent-Set Problem (Independent-Set)**
  - **Instance:** a graph \( G = (V, E) \) and a positive integer \( k \leq |V| \).
  - **Question:** is there an independent set of size \( \geq k \)?

- **Theorem:** The following are equivalent for \( G = (V, E) \) and a subset \( V' \) of \( V 

1. \( V' \) is a clique of \( G \).
2. \( V' \) is an independent set of \( \bar{G} \).
3. \( V-V' \) is a vertex cover of \( \bar{G} \).

- **Corollary:** Independent-Set is NP-complete.
A **hitting set** for a collection $C$ of subsets of a set $S$ is a subset $S'$ of $S$ such that $S'$ contains at least one element from each subset in $C$.

- $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $C = \{\{1\}, \{3, 5\}, \{4, 7, 8\}, \{5, 6\}\}$, $S'$ can be $\{1, 4, 5\}$, $\{1, 3, 4, 6\}$, etc.

**The Hitting-Set Problem (Hitting-Set)**
- **Instance:** Collection $C$ of subsets of a set $S$, positive integer $k$.
- **Question:** does $S$ contain a hitting set for $C$ of size $\leq k$?

**Hitting-Set is NP-Complete.**
- Restrict to Vertex-Cover by allowing only instances having $|c| = 2$ for all $c \in C$.
- Each set in $C \leftrightarrow$ edge; element in $S'$ $\leftrightarrow$ vertex cover.

**Proof by restriction** is the simplest, and perhaps the most frequently used technique.
- Other examples: Bounded Degree Spanning Tree, Directed HC, Longest Simple Cycle, etc.
Coping with NP-Complete/-Hard Problems

- **Approximation algorithms:**
  - Guarantee to be a fixed percentage away from the optimum.
- **Pseudo-polynomial time algorithms:**
  - E.g., dynamic programming for the 0-1 Knapsack problem.
- **Probabilistic algorithms:**
  - Assume some probabilistic distribution of the instances.
- **Randomized algorithms:**
  - Make use of a randomizer (random # generator) for operation.
- **Restriction:** Work on some special cases of the original problem.
  - E.g., the maximum independent set problem in circle graphs.
- **Exponential algorithms/Branch and Bound/Exhaustive search:**
  - Feasible only when the problem size is small.
- **Local search:**
  - Simulated annealing (hill climbing), genetic algorithms, etc.
- **Heuristics:** No formal guarantee of performance.
Exhaustive Search v.s. Branch and Bound

- TSP example

Backtracking/exhaustive search

Branch and bound
Simulated Annealing

Simulated Annealing Basics

• Non-zero probability for “up-hill” moves.
• Probability depends on
  1. magnitude of the “up-hill” movement
  2. total search time

\[
\text{Prob}(S \rightarrow S') = \begin{cases} 
1 & \text{if } \Delta C \leq 0 \quad / \ast \text{ “down-hill” moves } \ast / \\
\frac{\Delta C}{T} & \text{if } \Delta C > 0 \quad / \ast \text{ “up-hill” moves } \ast / 
\end{cases}
\]

• $\Delta C = \text{cost}(S') - \text{Cost}(S)$
• $T$: Control parameter (temperature)
• Annealing schedule: $T = T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$. 
Generic Simulated Annealing Algorithm

begin
1 Get an initial solution $S$;
2 Get an initial temperature $T > 0$;
3 while not yet “frozen” do
4   for $1 \leq i \leq P$ do
5      Pick a random neighbor $S'$ of $S$;
6      $\Delta \leftarrow \text{cost}(S') - \text{cost}(S);$ /* downhill move */
7      if $\Delta \leq 0$ then $S \leftarrow S'$/* uphill move */
8      if $\Delta > 0$ then $S \leftarrow S'$ with probability $e^{\frac{-\Delta}{T}}$; /* reduce temperature */
9      $T \leftarrow rT$; /* reduce temperature */
10 return $S$
11 end
Basic Ingredients for Simulated Annealing

- Analogy:

<table>
<thead>
<tr>
<th>Physical system</th>
<th>Optimization problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>configuration</td>
</tr>
<tr>
<td>energy</td>
<td>cost function</td>
</tr>
<tr>
<td>ground state</td>
<td>optimal solution</td>
</tr>
<tr>
<td>quenching</td>
<td>iterative improvement</td>
</tr>
<tr>
<td>careful annealing</td>
<td>simulated annealing</td>
</tr>
</tbody>
</table>

- Basic Ingredients for Simulated Annealing:
  - Solution space
  - Neighborhood structure
  - Cost function
  - Annealing schedule
B*-Tree Based Floorplanner Revisited

- Solution space: all feasible B*-trees
- Neighborhood structure: Find a new solution by
  - Op1: rotate a block
  - Op2: delete & insert
  - Op3: swap 2 nodes
- Cost function: area & wirelength
- Annealing schedule: $T_i = r^i T_0$, $r = 0.9$
Approximation Algorithms

• **Approximation algorithm**: An algorithm that returns near-optimal solutions.

• **Ratio (Performance) bound** $\rho(n)$: For any input size $n$, the cost $C$ of the solution produced by an approximation algorithm $\leq \rho(n)$ of the cost $C^*$ of an optimal solution:

$$\max \left( \frac{C}{C^*}, \frac{C^*}{C} \right) \leq \rho(n).$$

  - $\rho(n) \geq 1$.
  - An optimal algorithm has ratio bound 1.

• **Relative error bound** $\varepsilon(n)$:

$$\frac{|C - C^*|}{C^*} \leq \varepsilon(n).$$

  - $\varepsilon(n) \leq \rho(n) - 1$. 
Greedy Vertex Cover Algorithm Revisited

- **Greedy heuristic**: cover as many edges as possible (vertex with the maximum degree) at each stage and then delete the covered edges.

- **The greedy heuristic cannot always find an optimal solution!**
  - The vertex-cover problem is NP-complete.

- **The greedy heuristic cannot guarantee a constant performance bound.**

![Graph instance](image1.png)

A graph instance

![Vertex cover](image2.png)

A vertex cover of size 5 obtained by the greedy algorithm.

![Optimal vertex cover](image3.png)

A vertex cover of size 4 optimal solution!!
The Vertex-Cover Problem

Approx-Vertex-Cover(G)
1. $C \leftarrow \emptyset$
2. $E' \leftarrow E[G]$
3. while $E' \neq \emptyset$
4. let $(u, v)$ be an arbitrary edge of $E'$
5. $C \leftarrow C \cup \{u, v\}$
6. remove from $E'$ every edge incident on either $u$ or $v$
7. return $C$

- Time complexity: $O(E)$.

![Graphs showing vertex cover process](image-url)
Approx-Vertex-Cover Has a Ratio Bound of 2

Let $A$ denote the set of edges picked in line 4 $\Rightarrow |C| = 2|A|$.

Since no two edges in $A$ share an endpoint, no vertex in any cover is incident on more than one edge in $A$.

Thus, $|A| \leq |C^*|$ and $|C| \leq 2|C^*|$.

Recall: For a graph $G = (V, E)$, $V'$ is a minimum vertex cover $\iff V - V'$ is a maximum independent set.

Is there any polynomial-time approximation with a constant ratio bound for the maximum independent set problem?
Approximation Algorithm for TSP

Approx-TSP-Tour(G)
1. select a vertex \( r \in V[G] \) to be a “root” vertex;
2. grow a minimum spanning tree \( T \) for \( G \) from root \( r \) using MST-Prim(\( G, d, r \));
3. Let \( L \) be the list of vertices visited in a preorder tree walk of \( T \);
4. return the HC \( H \) that visits the vertices in the order \( L \).

- Time complexity: \( O(V \lg V) \).
Approx-TSP-Tour with Triangle Inequality

- Inter-city distances satisfy **triangle inequality** if for all vertices \( u, v, w \in V \), \( d(u, w) \leq d(u, v) + d(v, w) \).

- **Approx-TSP-Tour with triangle inequality** has a ratio bound of 2.
  - With triangle inequality: \( \text{cost}(H) \leq 2 \times \text{cost of MST} \).
  - Let \( H^* \) denote an optimal tour. \( H^* \) is formed by some tree plus an edge \( \Rightarrow \) \( \text{cost of MST} \leq \text{cost}(H^*) \).
  - Thus, \( \text{cost}(H) \leq 2 \times \text{cost}(H^*) \).
TSP without Triangle Inequality

- If P ≠ NP, there is no polynomial-time approximation algorithm with constant ratio bound ρ for the general TSP.
  - Suppose in contraction that there is such an algorithm A with a constant ρ. We will use A to solve HC in polynomial time.
  - Algorithm for HC
    1. Convert $G = (V, E)$ into an instance $I$ of TSP with cities $V$ (resulting in a complete graph $G' = (V, E')$):
       
       $d(u, v) = \begin{cases} 
       1, & \text{if } (u, v) \in E, \\
       \rho |V| + 1, & \text{otherwise.}
       \end{cases}$
    2. Run A on $I$.
    3. If the reported cost $\leq \rho |V|$, then return “Yes” (i.e., $G$ contains a tour that is an HC), else return “No.”
**Correctness**

- If $G$ has an HC: $G$ contains a tour of cost $|V|$ by picking edges in $E$, each with cost of 1.
- If $G$ does not have an HC: any tour of $G'$ must use some edge not in $E$, which has a total cost of $\geq (\rho |V| + 1) + (|V| - 1) > \rho |V|$.
- $A$ guarantees to return a tour of cost $\leq \rho \times$ cost of an optimal tour $\Rightarrow A$ returns a cost $\leq \rho |V|$ if $G$ contains an HC; $A$ returns a cost $> \rho |V|$, otherwise.