NP-Completeness Theory
Previously:

- Positive results: design efficient algorithm to solve problems

This part:

- Negative results: show some problems cannot be solved efficiently.

> Why study negative results?

- Practical: avoid wasting time on designing algs

- Theoretical: THE first problem of computer science [Turing '1936]
  - Leads to better algorithms
  - Fun!
Preliminary

- Computation model
  - Finite automata, Turing Machine, Quantum Computer.
  - RAM model

- Decision problem vs. Optimization Problem.
  - Decision problem: output is 0/1 (single bit)
    - Example: "Is there a \((s,t)\)-path in the graph \(G\) of length \(\leq L\)?"
  - Optimization problem: output is arbitrary
    - Example: "What is the shortest \((s,t)\)-path length in graph \(G\):"
• Optimization $\implies$ Decision
  - Trivial: E.g. if you know the shortest $(s,t)$-path length, then surely you can answer the question "Is it $\leq L$?"

• Decision $\implies$ Optimization
  - **Binary Search**: Guess a value $L$, then solve the decision prob "Is it $\leq L$?"
    - If "Yes", then try a smaller $L$
    - Otherwise, try a larger $L$.
  - Incurs an extra $O(\log n)$ factor on running time.
* "Efficient" = polynomial time, i.e. $n^O(1)$

- But why? A $O(n^{100})$-time alg doesn't seem to be useful.

* Ans:
  - For most realworld problems, if $\exists$ poly-time alg.
    then often the running time is $O(n^k)$ for some $k \leq 4$
  - A "Dichotomy phenomenon": often for natural problems
    - either we have a $O(n^k)$-time alg with some small $k$
    - or the best alg we have runs in $2^{O(n^k)}$-time
> Some hard problems

- Hamiltonian Path:
  
  Input: graph $G = (V, E)$, and $s, t \in V$
  
  Output: find a $(s, t)$-path that visits every vertex in $V$
  exactly once.

- Max Indep Set:
  
  Input: graph $G = (V, E)$
  
  Output: indep set $S \subseteq V$ with maximum size.
\( \textit{P}: \) polynomial-time solvable problems.

**Def:** The **complexity class** \( \textit{P} \) is the set of (decision) problems that can be solved in polynomial time.

- **Exmp:** MST, shortest-path, interval scheduling
- **Knapsack** is not known to be in \( \textit{P} \)
NP: Polynomial-time certifiable problems.

- Example: Hamiltonian Path prob. “Is there a (s,t)-HP in G?”

- Certification:
  - Given: Input graph G, and a (s,t)-path P
  - Goal: Verify if P is a HP

- Easy to solve.

- P is a certificate for the answer “Yes”

- Alg to certify P runs in poly(n) time.

⇒ certifier.
Def: The complexity class NP is the set of (decision) problems that can be certified in polynomial time.

Exmp: Hamiltonian Path, Max Indep Set, Vertex Cover, Knapsack.

- Max Indep Set: "Is the Max Indep Set has size $\geq L$?"
  - Certificate: an indep set of size $\geq L$
  - Certifier: Count the size of the set & check if $\forall u, v \in$ the given set, $(u,v) \notin E$

Observation: $P \subseteq NP$. 
The "P vs NP" problem

Question: Is $P = NP$?

- Common belief: $P \neq NP$
  
  "Assume $P \neq NP$, then $NP$ doesn't have poly-time alg."

- Consequence:
  
  - If $P \neq NP$: not much will change.
  
  - If $P = NP$: If one can check a sol efficiently, then one can find a sol efficiently.
Reduction:

- How do we compare the difficulty of two problems when we don't know the answer of \( P \equiv \text{NP} \).

E.g. How do we know MST is easier than HP, given that we cannot prove there's no poly-time alg for HP?

Reduction: If an alg A for prob X can be used to solve prob Y, then X is at least as hard as Y.
Consider two computation problems $X, Y$; if for any instance of problem $Y$, we can convert it to an instance of $X$ in poly-time, then we say $Y$ is poly-time reducible to $X$, denoted as $Y \leq_p X$.

**Corollary (positive):**

Given a poly-time alg $A$ that solves $X$, if $Y \leq_p X$, then $Y$ can also be solved in poly-time.
**Corollary (negative)**

If $Y \leq_p X$ and $Y$ cannot be solved in poly-time, then $X$ cannot be solved in poly-time either.

i.e. $Y \leq_p X$ implies $X$ is at least as hard as $Y$.

**Exmp:**

Suppose we can show $\text{MST} \leq_p \text{HP}$, then at least we know $\text{HP}$ is no easier than $\text{MST}$. 
Reduction: Converting \( Y \) to \( X \)

\[(G, L)\]  
Arbitrary \( Y \) inst.  
\[\downarrow\text{reduction: pre-proc}\]  
\[\downarrow\]  
\[(G', s, t)\]  
Corresponding \( X \) inst.  
\[\rightarrow\text{Alg A for } X\]  
Output for \( X \)  
\[\uparrow\text{reduction: post-proc}\]  
\[\downarrow\]  
\[T: a \text{ MST}\]  
Output for \( Y \)  
\[\uparrow\]  
\[p: a \text{ HP}\]
To show $Y \leq_p X$:

- Focusing on decision problem.
- Design a reduction alg that:

  1. show an "Yes" inst of $Y$ implies a "Yes" inst of $X$
  2. show an "No" inst of $Y$ implies a "No" inst of $X$

- Or equivalently: show a "Yes" inst of $X$ implies a "Yes" inst of $Y$
Example: $HP \leq_p HC$

Hamiltonian Cycle ($HC$)

Input: $G = (V, E)$, $s \in V$

Output: whether there is a cycle that starts from $s$, visits every other vertex exactly once, and ends at $s$. 

$HP$ inst: 

$G$

$HC$ inst: 

$G'$
NP-Complete problem.

A problem $X$ is NP-Complete if

1. $X \in NP$
2. $\forall Y \in NP$, $Y \leq_p X$

- $X$ is "the hardest problem" in $NP$
- Positive: A poly-time alg for $X$ will imply $P = NP$
- Negative: If you believe $P \neq NP$, and proved a problem to be NPC, then you can give up on designing poly-time algs for it.
Exmp.: HP, HC, Max Indep Set, Vertex Cover, Knapsack, Subset-Sum.

"Dichotomy":

Most natural NP problems are either NPC or P.

(There do exist some "intermediate" problems, e.g. Factoring; Graph Isomorphism.)
The first NPC problem: 3SAT

**Def (3-CNF):**

- **Boolean var** \( X_1, \ldots, X_n \in \{ \text{True}, \text{False} \} \)
- **Literals**: \( X_i \) or \( \neg X_i \)
- **Clause**: disjunction ("or") of \( \leq 3 \) literals
  
  e.g. \( X_3 \lor \neg X_4 \), \( X_1 \lor X_8 \lor X_9 \), \( \neg X_2 \lor \neg X_5 \)
- **3-CNF formula**: conjunction ("and") of clauses:
  
  \( (X_1 \lor X_2 \lor \neg X_3) \land (X_2 \lor X_4 \lor X_5) \land (\neg X_1 \lor \neg X_3 \lor X_6) \)
Prob (3-SAT)

Input: A 3-CNF formula involve var x_1, ..., x_n
Output: Whether the 3-CNF is satisfiable, i.e.
\[ \exists \text{ an assignment from } \{x_1, ..., x_n\} \text{ to } \{\text{True}, \text{False}\} \]
s.t. the 3-CNF evaluates to True.

- "Satisfied":
  1. every clause evals to True
  2. In each clause, at least one literal evals to true.

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

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>/</th>
<th>/</th>
<th>/</th>
<th>T</th>
</tr>
</thead>
</table>

- Thm [Cook' 1970s]: 3-SAT is NPC
$\text{3 SAT } \leq_p \text{ Indep Set. (i.e. Indep Set is NPC)}$

Example: 3 SAT inst

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

Indep Set inst:

$$(G, k)$$

"Is there an independent set in $G$ of size $\geq k$?"

Goal: construct an IndepSet inst $(G, k)$ based on $\phi$ s.t.

1. $\phi$ is satisfiable $\Rightarrow$ $\exists$ independent set of size $\geq k$ in $G$

2. $\phi$ is unsatisfiable $\Rightarrow$ $\forall$ independent set in $G$ is of size $< k$

or equivalently:

$\exists$ independent set of size $\geq k$ in $G$ $\Rightarrow$ $\phi$ is satisfiable
3-SAT

\( \chi_1 \leftarrow T, \chi_2 \leftarrow T \)
\( \chi_4 \leftarrow T, \chi_3 \leftarrow F \)

\( \phi = (\chi_1 \lor \neg \chi_2 \lor \neg \chi_3) \land \\
(\neg \chi_2 \lor \chi_3 \lor \chi_4) \land \\
(\neg \chi_1 \lor \neg \chi_3 \lor \chi_4) \)

Let \( n := \# \text{vbls in } \phi \)
\( m := \# \text{clauses} \)

\((G, k = m)\)

\text{Independent Set}

\text{clauses} \iff \text{group.}
1. \( \emptyset \) is satisfiable \( \Rightarrow \) \exists \text{ independent set } S \text{ of size } \geq k = m \text{ in } G \\
   \quad \cdot \text{ Every clause has } \geq 1 \text{ literal is true} \\
   \quad \cdot \text{ Pick exactly one literal that's true, add the corresponding vertex in } G \text{ to a set } S \\

\textbf{Claim: } S \text{ is a size-} m \text{ independent set in } G.

2. \exists \text{ independent set } S \text{ of size } \geq k = m \text{ in } G \Rightarrow \emptyset \text{ is satisfiable} \\

\textbf{Obs: } S \text{ contain exactly one vertex of each group} \\
   \quad \cdot \text{ For each } v \in S, \text{ let the corresponding literal be true,} \\

\textbf{Claim: } \text{ above assignment satisfies } \emptyset.
\[ \text{Indep} \text{Set} \leq_p \text{Vertex Cover (VC)} \]

\begin{minipage}{\textwidth}
\textbf{Prob Vertex Cover (VC)}:

\text{Input:} \text{ graph } G = (V, E), \text{ integer } k \\
\text{Output:} \text{ whether there's a VC of } G \text{ of size } \leq k
\end{minipage}

(Recall: a VC of G is some \( S \subseteq V \) st. \( \forall (u, v) \in E \), at least one of \( u, v \) are in \( S \).

\textbf{Obs:} If \( S \subseteq V \) is a VC in \( G \), then \( V \setminus S \) is an IS in \( G \).

\textbf{Pf:} Suppose \( V \setminus S \) is not an IS. Then, \( \exists (u, v) \in V \setminus S, \text{ s.t. } (u, v) \in E \). However, since \( S \) is a VC, at least one of \( u, v \) are in \( S \), which contradicts with \( u, v \in V \setminus S \). \( \Box \)
\[ \text{Pf of } \text{IndepSet} \leq_p \text{ Vertex Cover} \]

- A IS inst. \((G, k)\) with \(|V| = n\)

\[ \text{reduction} \]
\[ \rightarrow \text{ VC inst } (G, n-k), \text{ then} \]

1. \(\exists \text{ IS of size } \geq k \implies \exists \text{ VC of size } \leq n-k\)
   - Trivial by the previous OBS: take the complement of the IS

2. \(\exists \text{ VC of size } \leq n-k \implies \exists \text{ IS of size } \geq k\)
   - Same as above: take the complement of the VC.
Dealing with NP-hard problems

**Def:** Problem X is NP-hard if ∀Y ∈ NP, Y ≤_P X

*Note:* X itself is not required to be in NP.

- All NPC problems are NP-hard, but not vice versa.
- No hope for poly-time alg (assuming P ≠ NP)?
  - Faster exp-time alg
  - Approx alg
  - Solving for special cases
Faster exp-time alg

3-SAT:
- Brute-force: $O(2^n \cdot \text{poly}(n))$
- $2^n \rightarrow 1.344^n \rightarrow 1.34^n$
- In practice: up to 10000 variables can be solved in short time

Traveling Salesman Problem (i.e. the shortest Hamiltonian Cycle)
- Brute-force: $O(n! \cdot \text{poly}(n)) \rightarrow O(2^n \cdot \text{poly}(n))$
- In practice: Euclidean TSP with $\sim 100,000$ vertices.
Approx Alg.

Idea: ask for sub-optimal sols that are guaranteed not too far away from opt.

Exmp: Prob (Min Vertex Cover): given graph $G$, find the Vertex Cov in $G$ of minimum size.

- Can we find an VC of size $\leq \alpha \cdot \text{OPT}$ for some small $\alpha > 1$?
  Ans: there exists an alg outputs $VC \leq 2 \cdot \text{OPT}$ (2-approx)

- Can we find an IS of size $\geq \beta \cdot \text{OPT}$ for some big $\beta < 1$?
  Ans: unfortunately, no. No polytime alg can achieve $\beta > \Omega(\frac{1}{n})$
Solving for special cases

Example: Max-Indep Set is poly-time-solvable on

- Trees
- Bounded-treewidth graph
- Interval graphs