P: problem with a polynomial time algorithm

NP: polynomial time verification

\[ \Rightarrow \text{given a solution, verify that it is correct in polynomial time} \]

Class of decision problems \( \exists y \in \text{NP} \Rightarrow \) does there exist an optimal solution \( \leq y \)?

\[
\text{opt} - x \equiv \begin{cases} \text{d} - x \equiv \text{TSP} \\ \text{TSP} \end{cases}
\]

NP-complete \( \subseteq \) subset of NP

A problem \( L \) is NP-complete if

1) \( L \in \text{NP} \)

2) all other problems in NP, say \( L' \), can be reduced to \( L \)

\[ L' \Rightarrow L \quad \forall L' \in \text{NP} \]

("If \( L \) can be solved in \( P \) time then \( P = NP \)"

Q: how to show that a problem \( L \) is NPC/NS-complete?

[Approach 1]

look at all \( L' \in \text{NP} \)

and then show that \( L' \Rightarrow L \)

\[ g(I) \rightarrow x \rightarrow f(x) \rightarrow \text{poly time} \]

\[ \exists \text{a function } f, \text{ computable in polynomial time such that} \]

\( f(x) \) is a solution to \( L' \) if \( x \) is a solution to \( L \)

on an instance \( I \), there can also be
P \subseteq \text{NP} \checkmark

\text{P } \cap \text{NP C} = \varnothing \text{?} \ \text{do not know}

\text{P \neq \text{NP}}

\text{Approach 2: Is } L \text{ NP-complete?}

\text{Start from some known } \text{NP-complete} \text{ problem, say } L'

\text{Just show that } L' \rightarrow L \text{ and } L \text{ is in NP}

\forall L'' \in \text{NP}

\text{(NP-complete)}

\text{Just show this from the definition of NP-complete}

1^{st} \text{ NP-complete problem}

\text{Circuit SAT: circuit satisfiability}

\text{SAT problem: satisfiability problem}

\text{Input: a Boolean expression comprising of three logical operations (AND, OR, NOT)}
Input: a Boolean expression comprising of three logical operations (AND, OR, NOT)

Task: check whether there exists a satisfying assignment or not.

What is a Boolean formula/expression?

1) Variables: \( x, y, z \), etc.
   can be either 0 or 1

2) Literal: either a variable \( x \)
   or its negation \( \overline{x} \)

3) Clause: OR of literals (disjunction)
   \( (x \lor \overline{y} \lor z) \equiv x \lor (\neg y) \lor z \)

4) Expression: AND of clauses (conjunction)
   \( (x \lor \overline{y} \lor z) \land (y \lor w) \equiv \text{CNF} \)
   2 clauses
   length: sum of all the clause lengths
   \( 3 + 2 = 5 \)

Satisfying assignment

An assignment of \( 0 \) or \( 1 \) to each variable so that the Boolean expression is true

\[ I : (x \lor \overline{y} \lor z) \land (\overline{x} \lor \overline{y}) \land (y \lor z) \equiv \text{CNF} \]
AND \implies \text{ every clause has to be true}

OR

at least one literal

has to be true

\[\text{Algorithm 1 for SAT (try all possible assignments)}\]

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
<th>(\bar{x})</th>
<th>(\bar{y})</th>
<th>(\bar{z})</th>
<th>(x,y,z)</th>
<th>(\bar{x},\bar{y},\bar{z})</th>
<th>(x,y,\bar{z})</th>
<th>(\bar{x},\bar{y},z)</th>
<th>(x,\bar{y},\bar{z})</th>
<th>(\bar{x},y,z)</th>
<th>(xy,z)</th>
<th>(\bar{x},y,\bar{z})</th>
<th>(x,\bar{y},z)</th>
<th>(\bar{x},y,z)</th>
<th>(\text{Full expression})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\[2^N \text{ where } n = \#\text{ variables}\]

\[L: \ \text{length of expression}\]

\[O(L \cdot 2^N) \text{ complexity of naive / brute-force method}\]

\[\text{Exponential}\]

\[\text{Circuit SAT}\]

\[\Rightarrow \text{hardware version of SAT}\]

Any problem in \(\text{NP}\) has a \text{digital circuit} that encodes it.

Sure, if a solution to the \text{NP} problem \(y\) circuit \text{SAT} has a solution

\text{digital circuit must have a size polynomial in the input } I
digital circuit must have a size polynomial in the input \( I \)

Circuit SAT = NP complete

1) Circuit SAT \( \leq_{	ext{pol}} \text{ poly time} \) SAT
2) \( \forall L' \in \text{NP} \quad L' \rightarrow L \)

---

Hamilton Path: given a graph, does there exist a path that visits each vertex only once starting from \( s \) and ending at \( t \)

Hamilton Cycle/Rudreka Cycle: given a graph, does there exist a path starting from \( s \) that visits all other vertices only one and ends at \( s \)

It is known that Rudreka path \( \in \text{NPC} \), show that Rudreka cycle is also NPC

Show that \( \text{Rudreka Path} \rightarrow \text{Rudreka Cycle} \)

1) Rudreka cycle \( \in \text{NP} \)
   \[ O(1+V) \] time to verify that a solution is correct.
   a) length has to be \( n \), \( |V| = n \)
   \[ \text{given a solution} \]
   b) count all vertices \( \rightarrow O(n) \) time
2) a solution to path \( y \) solution to cycle

1) \( y \) is a \( s-t \) path that visits all vertices exactly once

\( y \) path then cycle

\( \Rightarrow \) show \( \exists \) a cycle that visits each vertex only once

2) show \( \exists \) a path \( \iff \Rightarrow \) a cycle

\[ \text{delete } x \]
\[ \text{then the remaining portion of the cycle is a path in } G \]

Show that TSP is \( NP \)-Complete

1) \( TSP \in NP \quad (\equiv TSP) \)

Given a solution \( s \), and a bound \( b \),

\[ O(|V|^3) \] to verify the cycle the tour, then check \( y \) if \( i \leq b \).

\[
\begin{bmatrix}
G: \text{complete graph, undirected, weighted} \\
\vdots \\
d_{ij} \text{ is the weight on edge } (i,j)
\end{bmatrix}
\]

\[
\begin{bmatrix}
y_0 \\
\vdots \\
y_n
\end{bmatrix}
\]
2) Hamilton/Rudra cycle $\Rightarrow$ TSP

\[ \subseteq \text{ NPC} \]

Rudra cycle (G)

\[ a \quad \rightarrow \quad b \quad \rightarrow \quad c \quad \rightarrow \quad d \]

\[ \Gamma_{SP}(G') \quad \rightarrow \]

\[ \begin{aligned}
& \text{original edge} \\
& \text{dij} = 1 \\
& \text{new edge} \\
& \text{dij} = 1 + c \\
& c \geq 1
\end{aligned} \]

\[ a) \text{ transformation is } O(n^2) \text{ in the worst case} \\
\text{with case} \]

\[ \text{a TSP tour in } G' \leq b = |V| = n \]

1) Rudra cycle $\Rightarrow$ TSP with $\text{cor} \leq n$

\[ s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n-1} \rightarrow s \]

\[ \text{Rudra cycle in } G \]

\[ \Rightarrow \text{ tour in } G', \text{ cor } n \text{ exactly n} \]

2) If there is no Rudra cycle, then $\exists$ a tour in $G'$ with $\text{cor} \leq n$ in $G$

\[ \downarrow \]

we cannot visit all nodes in $G$, exactly once, however $\exists$ tour in $G'$, but

the tour will have to use at least one of the new edges

$\Rightarrow$ cor of the tour is at least
Any tour has cost \( \leq n + c \) or cost \( \geq n + c \) \\
\[ \text{we can choose } c \text{ to be as large as we want} \]

\[ \text{gap} = c \]

Show that TSP has no polynomial time approximation.

\[ \text{opt} - \text{TSP} : \text{ minimization problem} \]

\[ \text{least cost on min cost tour} \]

Given \( G \), \( \text{opt}(G) \) is the value of the optimal tour on \( G \).

\[ \text{Given } A, \text{ approximation algo for TSP (polytime, greedy algo \to always pick the closest unvisited city)} \]

\[ A(G) \] is the cost of the solution using \( A \) on \( G \).

\[ \frac{\text{approx ratio for } A}{\text{opt}} \leq \frac{10^2}{10} = 10 \]

Connect Karp reducible problem to the gap \( c \).
Connect Rudraik cycle problem to the gap $C$

$G$

\[ C = ? \]

Assume that $\exists$ an approx algo with approx ratio $\alpha$

\[ \alpha = \max_{G} \frac{A(\ell)}{\text{opt}(\ell)} \]

Choose $C = n \cdot \alpha$

1) If $\alpha \in$ a Rudraik path, try tour of size $n$

\[ \Rightarrow \alpha = \frac{A(\ell)}{\text{opt}(\ell)} \Rightarrow A(\ell) = n \alpha \]

2) Then $\neg$ a Rudraik path, try $\ell + C = n + n \alpha$

\[ = n(1 + \alpha) \]

\[ \Rightarrow \alpha = \frac{A(\ell)}{n(1 + \alpha)} = A(\ell) \leq n \alpha (1 + \alpha) > n \alpha \]

$A(\ell)$ run in poly time

Contradiction unless $\ell = n \alpha$

\[ \Rightarrow \] poly time approx algo cannot exist