FORMAL LANGUAGES, AUTOMATA AND COMPUTATION

PROVING PROBLEMS NP-COMPLETE
Complexity Classes: P and NP

Polynomial time reducibility

Satisfiability Problem (SAT)
  - CNF, 3CNF Forms
  - 3SAT Problem

NP-Completeness

NP-Completeness of the SAT problem
  - Reduction from accepting computation histories of nondeterministic TMs to a SAT formula such that
    - A polynomial time NTM accepts \( w \) iff the corresponding SAT formula has a satisfying assignment.

3SAT is NP-Complete.
Showing Problems NP-complete

- Remember that in order to show a language $X$ to be NP-complete we need to show
  1. $X$ is in NP, and
  2. Every $Y$ in NP is polynomial time reducible to $X$,

- Part 1 is (usually) easy. You argue that there is polynomial time verifier for $X$, which, given a solution (certificate), will verify in polynomial time, that, it is a solution.

- For part 2, pick a known NP-complete problem $Z$
  1. We already know that all problems $Y$ in NP reduce to $Z$ in polynomial time.
  2. We produce a polynomial time algorithm that reduces all instances of $Z$ to some instance of $X$.
  3. So $Y \leq_P Z$ and $Z \leq_P X$ then $Y \leq_P X$. 
Theorem

CLIQUE is NP-complete.

Proof

1. We know 3SAT is NP-complete.
2. We know that $3SAT \leq_P CLIQUE$.
3. Hence $CLIQUE$ is NP-complete.
**The Vertex Cover Problem**

**Definition – Vertex Cover**
Given an undirected graph $G$, a vertex cover of $G$ is a subset of the nodes where every edge of $G$ touches one of those nodes.

$\text{VERTEX-COVER} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover} \}$. 
THE VERTEX COVER PROBLEM

THEOREM

VERTEX-COVER is NP-complete.

PROOF IDEA

- Show VERTEX-COVER is in NP.
  - Easy, the certificate is the vertex cover of size \( k \).
- We reduce an instance of 3SAT, \( \phi \), to a graph \( G \) and an integer \( k \) so that \( \phi \) is satisfiable whenever \( G \) has a vertex cover of size \( k \).
- We employ a concept called gadgets, groups of nodes with specific functions, in the graph.
  - Variable gadgets – representing literals
  - Clause gadgets – representing clauses
The Vertex Cover Problem

- Let $\phi$ be a 3-cnf formula with $m$ variables and $l$ clauses.
- We construct in polynomial-time, an instance of $\langle G, k \rangle$ where $k = m + 2l$.
  - For each variable $x$ in $\phi$, we add two nodes to $G$ labeled $x$ and $\overline{x}$, connected by an edge (variable gadget).
  - For every clause $(\ell_1 \lor \ell_2 \lor \ell_3)$ in $\phi$, we add 3 nodes labeled $\ell_1$, $\ell_2$ and $\ell_3$, with edges between every pair so that they form a triangle (clause gadget).
  - We add an edge between any two identically labelled nodes, one from a variable gadget and one from a clause gadget.
The Vertex Cover Problem

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

Variables and negations of variables

#nodes = 2(#variables) + 3(#clauses)
THE VERTEX COVER PROBLEM

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

Variables and negations of variables

\[\phi \text{ satisfiable} \implies \text{put "true" literals on top in vertex cover}\]

For each clause, pick a true literal and put other 2 in vertex cover
**THE VERTEX COVER PROBLEM**

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

Variables and negations of variables

\[k = 2(\#\text{clauses}) + (\#\text{variables})\]
THE HAMILTONIAN PATH PROBLEM

DEFINITION - HAMILTONIAN PATH

(Recall that) A Hamiltonian path in a directed graph $G$ is a directed path that goes through each node exactly once.

DEFINITION HAMILTONIAN PATH PROBLEM

$HAMPATH = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \}$. 

![Diagram of directed graph with labeled nodes and edges]
**The Hamiltonian Path Problem**

**Theorem**

*HAMPATH* is NP-complete.

**Proof Idea**

- We show $3SAT \leq_P HAMPATH$.
- We again use gadgets to represent the variables and clauses.
- For a given 3-cnf formula with $k$ clauses

$$\phi = (a_1 \lor b_1 \lor c_1) \land (a_2 \lor b_2 \lor c_2) \land \cdots \land (a_k \lor b_k \lor c_k)$$

where each $a_i$, $b_i$ or $c_i$ is a literal $x$ or $\overline{x}$. We have $l$ variables $x_1, x_2, \ldots x_l$. 
The Hamiltonian Path Problem

- 1-node gadgets for clauses
- Diamond-shaped gadgets for variables
The Hamiltonian Path Problem

- The middle spine in each diamond has $3k + 3$ nodes.
  - 3 nodes per clause + 1 to isolate them from the two literal nodes and 2 nodes on each side for the literals $x_i, \bar{x}_i$. 
The Hamiltonian Path Problem

- If $x_i$ appears in clause $c_j$, we add two edges from $j^{th}$ group in the spine to the $j^{th}$ clause node in the $i^{th}$ diamond.
The Hamiltonian Path Problem

- If $\overline{x}_i$ appears in clause $c_j$, we add two edges from $j^{th}$ group in the spine to the $j^{th}$ clause node in the $i^{th}$ diamond, but in the reverse direction.
The Hamiltonian Path Problem

Suppose $\phi$ is satisfiable.
Ignoring the clause nodes, we note that the Hamiltonian path
- starts at $s$
- goes through each diamond
- ends up at $t$.

In diamond $i$, it either goes left-to-right or right-to-left depending on the truth value of variable $x_i$. 
The clause nodes can be incorporated into the path using the detours we provided.

So if $x_i$ is true and is in clause $c_j$, we can take a detour to node for $c_j$ and back to the spine in the right direction.

Note that each detour is **optional** but we have to incorporate $c_j$ only once.
The Hamiltonian Path Problem

- The clause nodes can be incorporated into the path using the detours we provided.
- So if $\overline{x_i}$ is true and is in clause $c_j$, we can take a detour to node for $c_j$ and back to the spine in the reverse direction.
How about the reverse direction? If \( G \) has a Hamiltonian path then \( \phi \) has a satisfying assignment?

If the path is normal, that is, it goes through from \( s \) zigzagging through the diamonds, then clearly there is a satisfying assignment.

The following case can not happen!
The Undirected Hamiltonian Path

**Definition Hamiltonian Path Problem**

\[ UHAMPATH = \{ \langle G, s, t \rangle \mid G \text{ is an undirected graph with a Hamiltonian path from } s \text{ to } t \} \].

**Theorem**

*UHAMPATH* is NP-complete.

**Proof Idea**

- We reduce *HAMPATH* to *UHAMPATH*.
- All nodes except *s* and *t* in the directed graph *G*, map to 3 nodes in the undirected graph *G′*.
- *G* has a Hamiltonian path \( \Leftrightarrow *G′* has an undirected Hamiltonian path.
The Undirected Hamiltonian Path

**Theorem**

$UHAMPATH$ is NP-complete.

**Proof**

- $s$ in $G$ maps to $s^{\text{out}}$ in $G'$.
- $t$ in $G$ maps to $t^{\text{in}}$ in $G'$.
- Any other node $u_i$ maps to $u_i^{\text{in}}, u_i^{\text{mid}}, u_i^{\text{out}}$ in $G'$.
  - All arcs coming to $u_i$ in $G$ become edges incident on $u_i^{\text{in}}$ in $G'$.
  - All arcs going out from $u_i$ in $G$ become edges incident on $u_i^{\text{out}}$ in $G'$.
Note that if \( s, u_1, u_2, \ldots, u_k, t \) is a Hamiltonian path in \( G \) then

\[
\begin{align*}
&\quad s^{\text{out}}, u_1^{\text{in}}, u_1^{\text{mid}}, u_1^{\text{out}}, u_2^{\text{in}}, u_2^{\text{mid}}, u_2^{\text{out}}, \ldots, u_k^{\text{out}}, t^{\text{in}}
\end{align*}
\]

is a Hamiltonian path in \( G' \).

Any Hamiltonian path between \( s^{\text{out}} \) and \( t^{\text{in}} \), must go through the triple of nodes except for the start and end nodes.
**The Subset Sum Problem**

\[ \text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{ x_1, \ldots, x_m \} \text{ and for some } \{ y_1, \ldots, y_n \} \subseteq S, \sum y_i = t \} \]

**Theorem**

\text{SUBSET-SUM} is NP-complete.

**Proof Idea**

- We reduce 3SAT to an instance of the \text{SUBSET-SUM} problem with a set \( S \) and a bound \( t \),
  - so that if a formula \( \phi \) has a satisfying assignment,
  - then \( S \) has a subset \( T \) that adds to \( t \)
- We already know that \text{SUBSET-SUM} is in NP.
The Subset Sum Problem

Let $\phi$ be a formula with variables $x_1, x_2, \ldots, x_l$ and clauses $c_1, \ldots, c_k$.

We compute $m = 2 \times l + 2 \times k$ (large) numbers from $\phi$ and a bound $t$

Such that when we choose the numbers corresponding to the literals in the satisfying assignment, they add to $t$. 

### The Subset Sum Problem

For \( \phi = (x_1 \lor \overline{x_2} \lor x_3) \land (x_2 \lor x_3 \lor \cdots) \land \cdots \land (\overline{x_3} \lor \cdots \lor \cdots) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>\ldots</th>
<th>l</th>
<th>c_1</th>
<th>c_2</th>
<th>\ldots</th>
<th>c_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(z_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(y_2)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(z_2)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(y_3)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(z_3)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(\vdots)</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
</tr>
<tr>
<td>(y_l)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(z_l)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>(g_1)</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
</tr>
<tr>
<td>(h_1)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_2)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h_2)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_k)</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
</tr>
<tr>
<td>(h_k)</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>\ldots</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>\ldots</td>
<td>3</td>
</tr>
</tbody>
</table>
The Subset Sum Problem

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>i</th>
<th>c_1</th>
<th>c_2</th>
<th>...</th>
<th>c_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>z_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>y_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>z_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>y_3</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>z_3</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>y_l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>z_l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>g_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>h_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>g_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>h_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>g_k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>h_k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>...</td>
</tr>
</tbody>
</table>

- We choose one of the numbers \( y_i \) if \( x_i = 1 \), or \( z_i \) if \( x_i = 0 \).
- The left part of \( t \) will add up the right number.
- The right side columns will at least be 1 each.
- We take enough of the \( g \) and \( h \)'s to make them add up to 3.