NP-Completeness

Definition
Nondeterministic Polynomial Time (denoted by \( \text{NP} \)) is the class of all problems that have efficient certifiers (for YES instances).

Recall...
NP-Complete Problems

Definition
A problem \( X \) is said to be \( \text{NP-Complete} \) if
1. \( X \in \text{NP} \), and
2. (Hardness) For any \( Y \in \text{NP} \), \( Y \leq_p X \).

Recall...

Definition
\( \text{co-NP} \): class of all decision problems \( X \) s.t. \( \overline{X} \in \text{NP} \).
Examples: \( \text{UnSAT} \), \( \text{No-Hamiltonian-Cycle} \), \( \text{No-3-Colorable} \).
Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language \( L \) is **NP-Complete** ⇐⇒
   - \( L \) is in **NP**
   - for every \( L' \) in **NP**, \( L' \leq_p L \)
3. \( L \) is **NP-Hard** if for every \( L' \) in **NP**, \( L' \leq_p L \).
4. Cook-Level theorem...

Theorem (Cook-Levin)

**Circuit-SAT** is **NP-Complete**.

---

### Circuits

**Definition**

A circuit is a directed acyclic graph with

1. Input vertices (without incoming edges) labelled with \( 0, 1 \) or a distinct variable.
2. Every other vertex is labelled \( \lor, \land \) or \( \lnot \).
3. Single node output vertex with no outgoing edges.

---

### Cook-Levin Theorem

**Definition** (Circuit Satisfaction (**CSAT**).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value \( 1 \)?

**Theorem** (Cook-Levin)

**CSAT** is **NP-Complete**.

Need to show

1. **CSAT** is in **NP**.
2. every **NP** problem \( X \) reduces to **CSAT**.

---

### CSAT: Circuit Satisfaction

**Claim**

**CSAT** is in **NP**.

1. **Certificate**: Assignment to input variables.
2. **Certifier**: Evaluate the value of each gate in a topological sort of **DAG** and check the output gate value.
CSAT is NP-hard: Idea

1. Need to show that every NP problem X reduces to CSAT.
2. What does it mean that $X \in \text{NP}$?
3. $X \in \text{NP}$ implies that there are polynomials $p()$ and $q()$ and certifier/verifier program $C$ such that for every string $s$ the following is true:
   3.1 If $s$ is a YES instance ($s \in X$) then there is a proof $t$ of length $p(|s|)$ such that $C(s, t)$ says YES.
   3.2 If $s$ is a NO instance ($s \notin X$) then for every string $t$ of length at $p(|s|)$, $C(s, t)$ says NO.
   3.3 $C(s, t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time).

Reducing $X$ to CSAT

1. NP problem: a three tuple $\langle p, q, C \rangle$.
   - $C$: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem $X$: Given string $s$, is $s \in X$?
3. Equivalent:
   - $\exists$ proof $t$ of length $p(|s|)$ & $C(s, t)$ returns YES.
   - ...$C(s, t)$ runs in $q(|s|)$ time.
4. Reduce from $X$ to CSAT...
   Need an algorithm $\text{alg}$ that:
   4.1 takes $s$ (and $\langle p, q, C \rangle$).
   - Creates circuit $G$ in poly time in $|s|$.
   - $\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.
   4.2 $G$ is satisfiable
   $\iff \exists$ proof $t$ s.t. $C(s, t)$ returns YES.

Reducing $X$ to CSAT

1. Q: How do we reduce $X$ to CSAT?
2. Need algorithm $\text{alg}$ that:
   2.1 Input: $s$ (and $\langle p, q, C \rangle$).
   2.2 creates circuit $G$ in poly-time in $|s|$ ($\langle p, q, C \rangle$ fixed).
   2.3 $G$ satisfiable $\iff \exists$ proof $t$: $C(s, t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   3.1 Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)
   3.2 Known: $|t| \leq p(|s|)$ so express boolean string $t$ as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3.3 Asking if there is a proof $t$ that makes $C(s, t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make $G$ evaluate to true/YES.
Example: **Independent Set**

1. **Formal definition:**

   **Independent Set**

   **Instance:** $G = (V, E)$

   **Question:** Does $G = (V, E)$ have an Independent Set of size $\geq k$?

2. **Certificate:** Set $S \subseteq V$.

3. **Certifier:** Check $|S| \geq k$ and no pair of vertices in $S$ is connected by an edge.

4. **Q:** Formally, why is **Independent Set** in **NP**?

**Certifier for Independent Set**

Certifier $C(s, t)$ for **Independent Set**:

- if $(t_1 + t_2 + \ldots + t_n < k)$ then return NO
- else
  - for each $(i, j)$ do
    - if $(t_i \land t_j \land y_{i,j})$ then return NO
  - return YES

Example: **Independent Set**

Formally why is **Independent Set** in **NP**?

1. Input is a “binary” vector:

   $\langle n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k \rangle$

   encodes $\langle G, k \rangle$.

   1.1 $n$ is number of vertices in $G$

   1.2 $y_{i,j}$ is a bit which is 1 if edge $(i, j)$ is in $G$ and 0 otherwise (adjacency matrix representation)

   1.3 $k$: size of independent set.

2. **Certificate:** $t = t_1 t_2 \ldots t_n$.

   Interpretation: $t_i = 1$ if vertex $i$ is in independent set.

   ... 0 otherwise.

Example: **Independent Set**

Certifier circuit for Independent Set of size at least 2 for graph with 3 vertices

---

**Graph $G$ with $k = 2$**

- At least two vertices?
- Both ends of an edge?
Programs, Turing Machines and Circuits

1. **alg:** “program” that takes $f(|s|)$ steps on input string $s$.
2. **Questions:** What computer is used? What does *step* mean?
3. “Real” computers difficult to reason with mathematically:
   - instruction set is too rich
   - pointers and control flow jumps in one step
   - assumption that pointer to code fits in one word
4. Turing Machines:
   - simpler model of computation to reason with
   - can simulate real computers with polynomial slow down
   - all moves are *local* (head moves only one cell)

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine $M$
2. **Problem:** Given $M$, input $s$, $p$, $q$ decide if:
   - $\exists$ proof $t$ of length $\leq p(|s|)$
   - $M$ executed on the input $s$, $t$ halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   - computes $p(|s|)$ and $q(|s|)$.
   - As such, $M$:
     - uses at most $q(|s|)$ memory/tape cells.
     - can run for at most $q(|s|)$ time.
   - Simulates evolution of the states of $M$ and memory over time, using a big circuit.

Simulation of Computation via Circuit

1. $M$ state at time $\ell$: A string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of $M = \text{input string } s$, a guess $t$ of $p(|s|)$ “unknowns”, and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does $M$ stops in $q_{\text{accept}}$ with blank tape.
4. Build circuit $C_\ell$: Evaluates to YES $\iff$ transition of $M$ from time $\ell$ to time $\ell + 1$ valid. (Circuit of size $O(q(|s|))$).
5. $C = C_0 \land C_1 \land \cdots \land C_q(|s|)$.
   - Polynomial size!
6. Output of $C$ true $\iff$ sequence of states of $M$ is legal and leads to an accept state.

NP-Hardness of Circuit Satisfaction

Key Ideas in reduction:
1. Use TMs as the code for certifier for simplicity
2. Since $p()$ and $q()$ are known to $A$, it can set up all required memory and time steps in advance
3. Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time

Note: Above reduction can be done to SAT as well. Reduction to SAT was the original proof of Steve Cook.
**SAT is NP-Complete**

1. We have seen that SAT ∈ NP
2. To show NP-Hardness, we will reduce Circuit Satisfiability (CSAT) to SAT

Instance of CSAT (we label each node):

```
1, a ?, b ?, c 0, d ?, e
```

**Converting a circuit into a CNF formula**

Label the nodes

(A) Input circuit

(B) Label the nodes.

(C) Introduce var for each node.

(D) Write a sub-formula for each variable that is true if the var is computed correctly.

\[
x_k \quad \text{(Demand a sat' assignment!)}
\]

\[
x_k = x_i \land x_k
\]

\[
x_j = x_g \land x_h
\]

\[
x_i = \neg x_f
\]

\[
x_h = x_d \lor x_e
\]

\[
x_g = x_b \lor x_c
\]

\[
x_f = x_a \land x_b
\]

\[
x_d = 0
\]

\[
x_a = 1
\]
### Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula.

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$x_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_k = x_j \land x_j$</td>
<td>$\neg x_k \lor \neg x_j \lor x_j \lor x_j \lor x_k \lor \neg x_j \lor \neg x_j$</td>
</tr>
<tr>
<td>$x_j = x_g \land x_h$</td>
<td>$\neg x_j \lor x_g \lor x_j \lor x_h \lor x_h \lor \neg x_g \lor \neg x_h$</td>
</tr>
<tr>
<td>$x_i = \neg x_f$</td>
<td>$(x_i \lor \neg x_f) \lor (\neg x_i \lor \neg x_f)$</td>
</tr>
<tr>
<td>$x_h = x_d \lor x_e$</td>
<td>$(x_h \lor \neg x_d) \land (\neg x_h \lor x_d \lor x_e)$</td>
</tr>
<tr>
<td>$x_g = x_b \lor x_c$</td>
<td>$(x_g \lor \neg x_b) \land (x_g \lor \neg x_c) \land (\neg x_g \lor x_b \lor x_c)$</td>
</tr>
<tr>
<td>$x_f = x_a \land x_b$</td>
<td>$(\neg x_f \lor \neg x_a) \land (\neg x_f \lor x_b) \land (x_f \lor \neg x_a \lor \neg x_b)$</td>
</tr>
<tr>
<td>$x_d = 0$</td>
<td>$\neg x_d$</td>
</tr>
<tr>
<td>$x_a = 1$</td>
<td>$x_a$</td>
</tr>
</tbody>
</table>

We got a CNF formula that is satisfiable $\iff$ the original circuit is satisfiable.

### Reduction: CSAT ≤ P SAT

1. For each gate (vertex) $v$ in the circuit, create a variable $x_v$.
2. **Case $\neg$:** $v$ is labeled $\neg$ and has one incoming edge from $u$ (so $x_u = \neg x_u$). In SAT formula generate, add clauses $(x_u \lor x_v), (\neg x_u \lor \neg x_v)$. Observe that
   
   $x_v = \neg x_u$ is true $\iff (x_u \lor x_v)$ both true.

### Reduction: CSAT ≤ P SAT

Continued...

1. **Case $\lor$:** So $x_v = x_u \lor x_w$. In SAT formula generated, add clauses $(x_v \lor \neg x_u), (x_v \lor \neg x_w)$, and $(\neg x_v \lor x_u \lor x_w)$. Again, observe that

   
   $(x_v = x_u \lor x_w)$ is true $\iff (x_v \lor \neg x_u), (x_v \lor \neg x_w), (\neg x_v \lor x_u \lor x_w)$ all true.
Reduction: CSAT $\leq_P$ SAT

Continued...

1. Case $\land$: So $x_v = x_u \land x_w$. In SAT formula generated, add clauses $(\neg x_v \lor x_u), (\neg x_v \lor x_w)$, and $(x_v \lor \neg x_u \lor \neg x_w)$. Again observe that

$$x_v = x_u \land x_w \text{ is true } \iff (\neg x_v \lor x_u), \quad (\neg x_v \lor x_w), \quad \text{all true.}$$

$$x_v \lor \neg x_u \lor \neg x_w$$

Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$

0.1 Find values of all gates in $C$ under $a$

0.2 Give value of gate $v$ to variable $x_v$; call this assignment $a'$

0.3 $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$

0.1 Let $a'$ be the restriction of $a$ to only the input variables

0.2 Value of gate $v$ under $a'$ is the same as value of $x_v$ in $a$

0.3 Thus, $a'$ satisfies $C$

Theorem

SAT is NP-Complete.

Reduction: CSAT $\leq_P$ SAT

Continued...

1. If $v$ is an input gate with a fixed value then we do the following. If $x_v = 1$ add clause $x_v$. If $x_v = 0$ add clause $\neg x_v$

2. Add the clause $x_v$ where $v$ is the variable for the output gate

Proving that a problem $X$ is NP-Complete

1. To prove $X$ is NP-Complete, show

1.1 Show $X$ is in NP.

1.1.1 certificate/proof of polynomial size in input

1.1.2 polynomial time certifier $C(s, t)$

1.2 Reduction from a known NP-Complete problem such as CSAT or SAT to $X$

2. SAT $\leq_p X$ implies that every NP problem $Y \leq_p X$. Why?

Transitivity of reductions:

3. $Y \leq_p SAT$ and SAT $\leq_p X$ and hence $Y \leq_p X$. 

NP-Completeness via Reductions

1. What we currently know:
   1.1 CSAT is NP-Complete.
   1.2 CSAT $\leq_p$ SAT and SAT is in NP and hence SAT is NP-Complete.
   1.3 SAT $\leq_p$ 3SAT and hence 3SAT is NP-Complete.
   1.4 3SAT $\leq_p$ Independent Set (which is in NP) and hence Independent Set is NP-Complete.
   1.5 Vertex Cover is NP-Complete.
   1.6 Clique is NP-Complete.

2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-Complete.

3. A surprisingly frequent phenomenon!

Next...

Prove
- Hamiltonian Cycle Problem is NP-Complete.
- 3-Coloring is NP-Complete.
- Subset Sum.

Part I

NP-Completeness of Hamiltonian Cycle

Directed Hamiltonian Cycle

Input
Given a directed graph $G = (V, E)$ with $n$ vertices

Goal
Does $G$ have a Hamiltonian cycle?
- A Hamiltonian cycle is a cycle in the graph that visits every vertex in $G$ exactly once
Directed Hamiltonian Cycle is \textbf{NP-Complete}

- Directed Hamiltonian Cycle is in \textbf{NP}
  - \textbf{Certificate}: Sequence of vertices
  - \textbf{Certifier}: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- \textbf{Hardness}: Will prove...
  \(3\text{SAT} \leq \text{p} \text{ Directed Hamiltonian Cycle.}\)

\begin{itemize}
\item Reduction: First Ideas
  \begin{itemize}
  \item Viewing SAT: Assign values to \(n\) variables, and each clause has 3 ways in which it can be satisfied.
  \item Construct graph with \(2^n\) Hamiltonian cycles, where each cycle corresponds to some boolean assignment.
  \item Then add more graph structure to encode constraints on assignments imposed by the clauses.
  \end{itemize}
\end{itemize}

\begin{itemize}
\item Reduction
  \begin{enumerate}
  \item \textbf{3SAT} formula \(\varphi\) create a graph \(G_\varphi\) such that
    \begin{itemize}
    \item \(G_\varphi\) has a Hamiltonian cycle \iff \(\varphi\) is satisfiable
    \item \(G_\varphi\) should be constructible from \(\varphi\) by a polynomial time algorithm \(A\)
    \end{itemize}
  \item Notation: \(\varphi\) has \(n\) variables \(x_1, x_2, \ldots, x_n\) and \(m\) clauses \(C_1, C_2, \ldots, C_m\).
  \end{enumerate}
\end{itemize}

\begin{itemize}
\item The Reduction: Phase I
  \begin{itemize}
  \item Traverse path \(i\) from left to right \iff \(x_i\) is set to true.
  \item Each path has \(3(m + 1)\) nodes where \(m\) is number of clauses in \(\varphi\); nodes numbered from left to right (1 to \(3m + 3\)).
  \end{itemize}
\end{itemize}
The Reduction: Phase II

- Add vertex $c_j$ for clause $C_j$. $c_j$ has edge from vertex $3j$ and to vertex $3j + 1$ on path $i$ if $x_i$ appears in clause $C_j$, and has edge from vertex $3j + 1$ and to vertex $3j$ if $\neg x_i$ appears in $C_j$.

In the next lecture...
Correctness proof of the above reduction, and more [NPC] problems.