Chapter 3

NP Completeness

3.1 NP Completeness

3.1.0.1 Certifiers

Definition 3.1.1. An algorithm $C(\cdot, \cdot)$ is a certifier for problem $X$ if for every $s \in X$ there is some string $t$ such that $C(s, t) = "yes"$, and conversely, if for some $s$ and $t$, $C(s, t) = "yes"$ then $s \in X$.

The string $t$ is called a certificate or proof for $s$.

Definition 3.1.2 (Efficient Certifier.). A certifier $C$ is an efficient certifier for problem $X$ if there is a polynomial $p(\cdot)$ such that for every string $s$, we have that

(A) $|t| \leq p(|s|)$,
(B) $C(s, t) = "yes"$,
(C) and $C$ runs in polynomial time.

3.1.0.2 NP-Complete Problems

Definition 3.1.3. A problem $X$ is said to be NP-Complete if

(A) $X \in \text{NP}$, and
(B) (Hardness) For any $Y \in \text{NP}$, $Y \leq_p X$.

3.1.0.3 Solving NP-Complete Problems

Proposition 3.1.4. Suppose $X$ is NP-Complete. Then $X$ can be solved in polynomial time if and only if $P = \text{NP}$.

Proof:

$\Rightarrow$ Suppose $X$ can be solved in polynomial time

(A) Let $Y \in \text{NP}$. We know $Y \leq_p X$.
(B) We showed that if $Y \leq_p X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
Thus, every problem \( Y \in \text{NP} \) is such that \( Y \in P; \text{NP} \subseteq P \).

(D) Since \( P \subseteq \text{NP} \), we have \( P = \text{NP} \).

\( \Rightarrow \) Since \( P = \text{NP} \), and \( X \in \text{NP} \), we have a polynomial time algorithm for \( X \).

3.1.0.4 NP-Hard Problems

Definition 3.1.5. A problem \( X \) is said to be NP-Hard if

(A) (Hardness) For any \( Y \in \text{NP} \), we have that \( Y \leq_P X \).

An NP-Hard problem need not be in \( \text{NP} \)!
Example: Halting problem is NP-Hard (why?) but not NP-Complete.

3.1.0.5 Consequences of proving NP-Completeness

If \( X \) is NP-Complete

(A) Since we believe \( P \neq \text{NP} \),

(B) and solving \( X \) implies \( P = \text{NP} \).

\( X \) is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for \( X \).

(This is proof by mob opinion — take with a grain of salt.)

3.1.1 Preliminaries

3.1.1.1 NP-Complete Problems

Question Are there any problems that are NP-Complete? Answer Yes! Many, many problems are NP-Complete.

3.1.1.2 Circuits

Definition 3.1.6. A circuit is a directed acyclic graph with

3.1.2 Cook-Levin Theorem

3.1.2.1 Cook-Levin Theorem

Definition 3.1.7 (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 3.1.8 (Cook-Levin). CSAT is NP-Complete.

Need to show

(A) CSAT is in \( \text{NP} \).

(B) every NP problem \( X \) reduces to CSAT.
3.1.2.2 **CSAT**: Circuit Satisfaction

Claim 3.1.9. **CSAT** is in **NP**.

(A) **Certificate**: Assignment to input variables.
(B) **Certifier**: Evaluate the value of each gate in a topological sort of **DAG** and check the output gate value.

3.1.2.3 **CSAT** is NP-hard: Idea

Need to show that every **NP** problem \(X\) reduces to **CSAT**.

What does it mean that \(X \in \text{NP}\)?

\(X \in \text{NP}\) implies that there are polynomials \(p()\) and \(q()\) and certifier/verifier program \(C\) such that for every string \(s\) the following is true:

(A) 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.  
(B) If \(s\) is a NO instance \((s \not\in X)\) then for every string \(t\) of length at \(p(|s|)\), \(C(s, t)\) says NO.  
(C) \(C(s, t)\) runs in time \(q(|s| + |t|)\) time (hence polynomial time).

3.1.2.4 Reducing \(X\) to **CSAT**

\(X\) is in **NP** means we have access to \(p(), q(), C(\cdot, \cdot)\).

What is \(C(\cdot, \cdot)\)? It is a program or equivalently a Turing Machine!

How are \(p()\) and \(q()\) given? As numbers.

Example: if 3 is given then \(p(n) = n^3\).

Thus an **NP** problem is essentially a three tuple \(\langle p, q, C \rangle\) where \(C\) is either a program or a TM.

3.1.2.5 Reducing \(X\) to **CSAT**

Thus an **NP** problem is essentially a three tuple \(\langle p, q, C \rangle\) where \(C\) is either a program or a TM.

**Problem X**: Given string \(s\), is \(s \in X\)?

Same as the following: is there a proof \(t\) of length \(p(|s|)\) such that \(C(s, t)\) says YES.

How do we reduce \(X\) to **CSAT**? Need an algorithm \(\mathcal{A}\) that

(A) takes \(s\) (and \(\langle p, q, C \rangle\)) and creates a circuit \(G\) in polynomial time in \(|s|\) (note that \(\langle p, q, C \rangle\) are fixed).

(B) \(G\) is satisfiable if and only if there is a proof \(t\) such that \(C(s, t)\) says YES.

3.1.2.6 Reducing \(X\) to **CSAT**

How do we reduce \(X\) to **CSAT**? Need an algorithm \(\mathcal{A}\) that

(A) takes \(s\) (and \(\langle p, q, C \rangle\)) and creates a circuit \(G\) in polynomial time in \(|s|\) (note that \(\langle p, q, C \rangle\) are fixed).

(B) \(G\) is satisfiable if and only if there is a proof \(t\) such that \(C(s, t)\) says YES.

**Simple but Big Idea**: Programs are essentially the same as Circuits!

(A) Convert \(C(s, t)\) into a circuit \(G\) with \(t\) as unknown inputs (rest is known including \(s\))

(B) We know that \(|t| = p(|s|)\) so express boolean string \(t\) as \(p(|s|)\) variables \(t_1, t_2, \ldots, t_k\) where \(k = p(|s|)\).

(C) 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.
3.1.2.7 Example: Independent Set

(A) Problem: Does $G = (V, E)$ have an Independent Set of size $\geq k$?
   (A) Certificate: Set $S \subseteq V$.
   (B) Certifier: Check $|S| \geq k$ and no pair of vertices in $S$ is connected by an edge.
      Formally, why is Independent Set in NP?

3.1.2.8 Example: Independent Set

Formally why is Independent Set in NP?
   (A) Input: $< n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k >$ encodes $< G, k >$.
      (A) $n$ is number of vertices in $G$
      (B) $y_{i,j}$ is a bit which is 1 if edge $(i, j)$ is in $G$ and 0 otherwise (adjacency matrix representation)
      (C) $k$ is size of independent set.
   (B) Certificate: $t = t_1 t_2 \ldots t_n$. Interpretation is that $t_i$ is 1 if vertex $i$ is in the independent set, 0 otherwise.

3.1.2.9 Certifier for Independent Set

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

\begin{verbatim}
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
\end{verbatim}

3.1.3 Example: Independent Set

3.1.3.1 A certifier circuit for Independent Set

Figure 3.1: Graph $G$ with $k = 2$

3.1.3.2 Programs, Turing Machines and Circuits

Consider “program” $A$ that takes $f(|s|)$ steps on input string $s$. 
Question: What computer is the program running on and what does step mean?

Real computers difficult to reason with mathematically because
(A) instruction set is too rich
(B) pointers and control flow jumps in one step
(C) assumption that pointer to code fits in one word

Turing Machines
(A) simpler model of computation to reason with
(B) can simulate real computers with polynomial slow down
(C) all moves are local (head moves only one cell)

3.1.3.3 Certifiers that at TMs

Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine $M$

Problem: Given $M$, input $s$, $p$, $q$ decide if there is a proof $t$ of length $p(|s|)$ such that $M$ on $s, t$ will halt in $q(|s|)$ time and say YES.

There is an algorithm $A$ that can reduce above problem to **CSAT** mechanically as follows.
(A) $A$ first computes $p(|s|)$ and $q(|s|)$.
(B) Knows that $M$ can use at most $q(|s|)$ memory/tape cells
(C) Knows that $M$ can run for at most $q(|s|)$ time
(D) Simulates the evolution of the state of $M$ and memory over time using a big circuit.

3.1.3.4 Simulation of Computation via Circuit

(A) Think of $M$’s state at time $\ell$ as a string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
(B) At time 0 the state of $M$ consists of input string $s$ a guess $t$ (unknown variables) of length $p(|s|)$ and rest $q(|s|)$ blank symbols.
(C) At time $q(|s|)$ we wish to know if $M$ stops in $q_{accept}$ with say all blanks on the tape.
(D) We write a circuit $C_\ell$ which captures the transition of $M$ from time $\ell$ to time $\ell + 1$.
(E) Composition of the circuits for all times 0 to $q(|s|)$ gives a big (still poly) sized circuit $C$
(F) The final output of $C$ should be true if and only if the entire state of $M$ at the end leads to an accept state.

3.1.3.5 NP-Hardness of Circuit Satisfaction

Key Ideas in reduction:
(A) Use TMs as the code for certifier for simplicity
(B) Since $p()$ and $q()$ are known to $A$, it can set up all required memory and time steps in advance
(C) 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.

3.1.4 Other NP Complete Problems

3.1.4.1 **SAT** is NP-Complete

(A) We have seen that **SAT** $\in$ NP
(B) To show NP-Hardness, we will reduce Circuit Satisfiability (**CSAT**) to **SAT**

Instance of **CSAT** (we label each node):
3.1.5 Converting a circuit into a CNF formula

3.1.5.1 Label the nodes

3.1.6 Converting a circuit into a CNF formula

3.1.6.1 Introduce a variable for each node
3.1.7 Converting a circuit into a CNF formula

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

\[ x_k \quad \text{(Demand a sat’ assignment!)} \]
\[ x_k = x_i \land x_j \]
\[ x_j = x_g \land x_h \]
\[ x_i = \neg x_f \]
\[ x_h = x_d \lor x_e \]
\[ x_g = x_h \lor x_c \]
\[ x_f = x_a \land x_b \]
\[ x_d = 0 \]
\[ x_a = 1 \]

(C) Introduce var for each node.

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

3.1.8 Converting a circuit into a CNF formula

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

3.1.9 Converting a circuit into a CNF formula

3.1.9.1 Take the conjunction of all the CNF sub-formulas

\[ x_k \land \neg x_k \land \neg x_i \land \neg x_j \land \neg x_g \land \neg x_h \land \neg x_f \land \neg x_a \land \neg x_b \land \neg x_d \land x_a \]

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.

3.1.9.2 Reduction: \( \text{CSAT} \leq_P \text{SAT} \)

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

$$x_v = \neg x_u \text{ is true } \iff (x_u \lor x_v) \text{ and } (\neg x_u \lor \neg x_v) \text{ both true.}$$

3.1.10 Reduction: CSAT $\leq_P$ SAT

3.1.10.1 Continued...

(A) 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

$$\left(x_v = x_u \lor x_w \right) \text{ is true } \iff \begin{cases} (x_u \lor \neg x_u), \\ (x_v \lor \neg x_w), \quad \text{all true.} \\ (\neg x_v \lor x_u \lor x_w) \end{cases}$$

3.1.11 Reduction: CSAT $\leq_P$ SAT

3.1.11.1 Continued...

(A) 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

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

3.1.12 Reduction: CSAT $\leq_P$ SAT

3.1.12.1 Continued...

(A) 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 $\neg x_v$

(B) Add the clause $x_v$ where $v$ is the variable for the output gate

3.1.12.2 Correctness of Reduction

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

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

(A) Find values of all gates in $C$ under $a$

(B) Give value of gate $v$ to variable $x_v$; call this assignment $a'$

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

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

(A) Let $a'$ be the restriction of $a$ to only the input variables

(B) Value of gate $v$ under $a'$ is the same as value of $x_v$ in $a$

(C) Thus, $a'$ satisfies $C$

Theorem 3.1.10. SAT is NP-Complete.
3.1.12.3 Proving that a problem \( X \) is NP-Complete

To prove \( X \) is \textbf{NP-Complete}, show

(A) Show \( X \) is in \textbf{NP}.
   
   (A) certificate/proof of polynomial size in input
   
   (B) polynomial time certifier \( C(s,t) \)

(B) Reduction from a known \textbf{NP-Complete} problem such as \textbf{CSAT} or \textbf{SAT} to \( X \)
   
   SAT \( \leq_p X \) implies that every \textbf{NP} problem \( Y \leq_p X \). Why?

Transitivity of reductions:

\( Y \leq_p SAT \) and \( SAT \leq_p X \) and hence \( Y \leq_p X \).

3.1.12.4 NP-Completeness via Reductions

(A) \textbf{CSAT} is \textbf{NP-Complete}.

(B) \textbf{CSAT} \( \leq_p \textbf{SAT} \) and \textbf{SAT} is in \textbf{NP} and hence \textbf{SAT} is \textbf{NP-Complete}.

(C) \textbf{SAT} \( \leq_p \textbf{3-SAT} \) and hence 3-SAT is \textbf{NP-Complete}.

(D) \textbf{3-SAT} \( \leq_p \textbf{Independent Set} \) (which is in \textbf{NP}) and hence \textbf{Independent Set} is \textbf{NP-Complete}.

(E) \textbf{Vertex Cover} is \textbf{NP-Complete}.

(F) \textbf{Clique} is \textbf{NP-Complete}.

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

A surprisingly frequent phenomenon!