Chapter 22

NP Completeness and Cook-Levin Theorem

CS 473: Fundamental Algorithms, Spring 2013
April 17, 2013

22.1 NP

22.1.0.1 P and NP and Turing Machines

(A) **P**: set of decision problems that have polynomial time algorithms.
(B) **NP**: set of decision problems that have polynomial time non-deterministic algorithms.

**Question**: What is an algorithm? Depends on the model of computation!

What is our model of computation?

Formally speaking our model of computation is Turing Machines.

22.1.0.2 Turing Machines: Recap

![Turing Machine Diagram]

(A) Infinite tape.
(B) Finite state control.
(C) Input at beginning of tape.
(D) Special tape letter “blank” ⊥.
(E) Head can move only one cell to left or right.

Prabhakaran-Viswanathan CS373
22.1.0.3 Turing Machines: Formally

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:

(A) $Q$ is set of states in finite control
(B) $q_0$ start state, $q_{\text{accept}}$ is accept state, $q_{\text{reject}}$ is reject state
(C) $\Sigma$ is input alphabet, $\Gamma$ is tape alphabet (includes $\sqcup$)
(D) $\delta : Q \times \Gamma \rightarrow \{L, R\} \times \Gamma \times Q$ is transition function

(A) $\delta(q, a) = (q', b, L)$ means that $M$ in state $q$ and head seeing $a$ on tape will move to state $q'$ while replacing $a$ on tape with $b$ and head moves left.

$L(M)$: language accepted by $M$ is set of all input strings $s$ on which $M$ accepts; that is:

(A) TM is started in state $q_0$.
(B) Initially, the tape head is located at the first cell.
(C) The tape contain $s$ on the tape followed by blanks.
(D) The TM halts in the state $q_{\text{accept}}$.

22.1.0.4 $P$ via TMs

Definition 22.1.1. $M$ is a polynomial time TM if there is some polynomial $p(\cdot)$ such that on all inputs $w$, $M$ halts in $p(|w|)$ steps.

Definition 22.1.2. $L$ is a language in $P$ iff there is a polynomial time TM $M$ such that $L = L(M)$.

22.1.0.5 NP via TMs

Definition 22.1.3. $L$ is an NP language iff there is a non-deterministic polynomial time TM $M$ such that $L = L(M)$.

Non-deterministic TM: each step has a choice of moves

(A) $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$.

(A) Example: $\delta(q, a) = \{(q_1, b, L), (q_2, c, R), (q_3, a, R)\}$ means that $M$ can non-deterministically choose one of the three possible moves from $(q, a)$.

(B) $L(M)$: set of all strings $s$ on which there exists some sequence of valid choices at each step that lead from $q_0$ to $q_{\text{accept}}$.

22.1.0.6 Non-deterministic TMs vs certifiers

Two definition of NP:

(A) $L$ is in NP iff $L$ has a polynomial time certifier $C(\cdot, \cdot)$.
(B) $L$ is in NP iff $L$ is decided by a non-deterministic polynomial time TM $M$.

Claim 22.1.4. Two definitions are equivalent.

Why?

Informal proof idea: the certificate $t$ for $C$ corresponds to non-deterministic choices of $M$ and vice-versa.

In other words $L$ is in NP iff $L$ is accepted by a NTM which first guesses a proof $t$ of length poly in input $|s|$ and then acts as a deterministic TM.
22.1.0.7 Non-determinism, guessing and verification

(A) A non-deterministic machine has choices at each step and accepts a string if there exists a set of choices which lead to a final state.

(B) Equivalently the choices can be thought of as guessing a solution and then verifying that solution. In this view all the choices are made a priori and hence the verification can be deterministic. The “guess” is the “proof” and the “verifier” is the “certifier”.

(C) We reemphasize the asymmetry inherent in the definition of non-determinism. Strings in the language can be easily verified. No easy way to verify that a string is not in the language.

22.1.0.8 Algorithms: TMs vs RAM Model

Why do we use TMs some times and RAM Model other times?

(A) TMs are very simple: no complicated instruction set, no jumps/pointers, no explicit loops etc.

   (A) Simplicity is useful in proofs.
   (B) The “right” formal bare-bones model when dealing with subtleties.

(B) RAM model is a closer approximation to the running time/space usage of realistic computers for reasonable problem sizes

   (A) Not appropriate for certain kinds of formal proofs when algorithms can take super-polynomial time and space

22.2 Cook-Levin Theorem

22.2.1 Completeness

22.2.1.1 “Hardest” Problems

Question What is the hardest problem in \( NP \)? How do we define it? Towards a definition

(A) Hardest problem must be in \( NP \).

(B) Hardest problem must be at least as “difficult” as every other problem in \( NP \).

22.2.1.2 NP-Complete Problems

Definition 22.2.1. A problem \( X \) is said to be \( NP \)-Complete if

(A) \( X \) ∈ \( NP \), and

(B) (Hardness) For any \( Y \) ∈ \( NP \), \( Y \leq_p X \).

22.2.1.3 Solving NP-Complete Problems

Proposition 22.2.2. Suppose \( X \) is \( NP \)-Complete. Then \( X \) can be solved in polynomial time if and only if \( P = NP \).

Proof:

⇒ Suppose \( X \) can be solved in polynomial time
Let $Y \in \text{NP}$. We know $Y \leq_P X$.

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 $Y \in P; NP \subseteq P$.

Since $P \subseteq \text{NP}$, we have $P = \text{NP}$.

\[\text{If } P = \text{NP}, \text{ and } X \in \text{NP}, \text{ we have a polynomial time algorithm for } X.\]

\[\text{NP-Hard Problems}\]

\textbf{Definition 22.2.3.} A problem $X$ is said to be \textbf{NP-Hard} if

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

An \textbf{NP-Hard} problem need not be in \textbf{NP}!

\textbf{Example:} Halting problem is \textbf{NP-Hard} (why?) but not \textbf{NP-Complete}.

\section*{Consequences of proving NP-Completeness}

If $X$ is \textbf{NP-Complete}

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

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

$X$ is \textbf{unlikely} to be efficiently solvable.

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

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

\section*{Preliminaries}

\subsection*{NP-Complete Problems}

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

\subsection*{Circuits}

\textbf{Definition 22.2.4.} A \textit{circuit} is a directed acyclic graph with

\[\text{Output: } X\]

\[\text{Inputs: } 1, Y, ?, 0, ?\]
22.2.3 Cook-Levin Theorem

22.2.3.1 Cook-Levin Theorem

**Definition 22.2.5 (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 22.2.6 (Cook-Levin).** CSAT is NP-Complete.

Need to show

(A) **CSAT** is in NP.

(B) every NP problem X reduces to CSAT.

22.2.3.2 CSAT: Circuit Satisfaction

**Claim 22.2.7.** 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.

22.2.3.3 CSAT is NP-hard: Idea

Need to show that every NP problem X reduces to CSAT.

What does it mean that $X \in NP$?

$X \in NP$ 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).

22.2.3.4 Reducing X to CSAT

$X$ is in NP means we have access to $p(), q(), C(·, ·)$.

What is $C(·, ·)$? 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.

22.2.3.5 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 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 $A$ 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.

22.2.3.6 Reducing $X$ to CSAT

How do we reduce $X$ to CSAT? Need an algorithm $A$ 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.

22.2.3.7 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?

22.2.3.8 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.

22.2.3.9 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
```
22.2.4 Example: Independent Set

22.2.4.1 A certifier circuit for Independent Set

Figure 22.1: Graph $G$ with $k = 2$

22.2.4.2 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)

22.2.4.3 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.

22.2.4.4 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 \( q_{\text{accept}} \) 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.

22.2.4.5 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.

22.2.5 Other NP Complete Problems

22.2.5.1 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):
22.2.6  Converting a circuit into a CNF formula

22.2.6.1  Label the nodes

(A) Input circuit  (B) Label the nodes.

22.2.7  Converting a circuit into a CNF formula

22.2.7.1  Introduce a variable for each node

(B) Label the nodes.  (C) Introduce var for each node.
22.2.8 Converting a circuit into a CNF formula

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

![Circuit Diagram]

\( x_k \) (Demand a sat' assignment!)
\( x_k = x_i \land x_j \)
\( x_j = x_g \land x_h \)
\( x_i = \neg x_f \)
\( x_h = x_d \lor x_e \)
\( x_g = x_b \lor x_c \)
\( x_f = x_a \land x_b \)
\( x_d = 0 \)
\( x_a = 1 \)

(C) Introduce var for each node.

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

22.2.9 Converting a circuit into a CNF formula

22.2.9.1 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 )</td>
<td>( (\neg x_k \lor x_i) \land (\neg x_k \lor x_j) \land (x_k \lor \neg x_i \lor \neg x_j) )</td>
</tr>
<tr>
<td>( x_j = x_g \land x_h )</td>
<td>( (\neg x_j \lor x_g) \land (\neg x_j \lor x_h) \land (x_j \lor \neg x_g \lor \neg x_h) )</td>
</tr>
<tr>
<td>( x_i = \neg x_f )</td>
<td>( (x_i \lor x_f) \land (\neg x_i \lor x_f) )</td>
</tr>
<tr>
<td>( x_h = x_d \lor x_e )</td>
<td>( (x_h \lor \neg x_d) \land (x_h \lor \neg x_e) \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 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>
22.2.10 Converting a circuit into a CNF formula

22.2.10.1 Take the conjunction of all the CNF sub-formulas

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

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

22.2.10.2 Reduction: CSAT \( \leq P \) SAT

(A) For each gate (vertex) \( v \) in the circuit, create a variable \( x_v \)

(B) 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 \( (x_v \lor x_u), (\neg x_u \lor \neg x_v) \). Observe that

\[ x_v = \neg x_u \text{ is true } \iff (x_v \lor x_u), (\neg x_u \lor \neg x_v) \text{ both true.} \]

22.2.11 Reduction: CSAT \( \leq P \) SAT

22.2.11.1 Continued...

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

\[ (x_v = x_u \lor x_w) \text{ is true } \iff (x_v \lor \neg x_u), (x_v \lor \neg x_w), (\neg x_v \lor x_u \lor x_w) \text{ all true.} \]

22.2.12 Reduction: CSAT \( \leq P \) SAT

22.2.12.1 Continued...

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

\[ x_v = x_u \land x_w \text{ is true } \iff (\neg x_v \lor x_u), (\neg x_v \lor x_w), (x_v \lor \neg x_u \lor \neg x_w) \text{ all true.} \]
22.2.13 Reduction: CSAT \( \leq_P \) SAT

22.2.13.1 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.

22.2.13.2 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 22.2.8. SAT is NP-Complete.

22.2.13.3 Proving that a problem \( X \) is NP-Complete

To prove \( X \) is NP-Complete, show
(A) Show \( X \) is in NP.
   (A) certificate/proof of polynomial size in input
   (B) polynomial time certifier \( C(s, t) \)
(B) Reduction from a known NP-Complete problem such as CSAT or SAT to \( X \)
\( \text{SAT} \leq_P X \) implies that every NP problem \( Y \leq_P X \). Why?

Transitivity of reductions:
\( Y \leq_P \text{SAT} \) and \( \text{SAT} \leq_P X \) and hence \( Y \leq_P X \).

22.2.13.4 NP-Completeness via Reductions

(A) CSAT is NP-Complete.
(B) CSAT \( \leq_P \) SAT and SAT is in NP and hence SAT is NP-Complete.
(C) SAT \( \leq_P \) 3-SAT and hence 3-SAT is NP-Complete.
(D) 3-SAT \( \leq_P \) Independent Set (which is in NP) and hence Independent Set is NP-Complete.
(E) Vertex Cover is NP-Complete.
(F) Clique is NP-Complete.

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

A surprisingly frequent phenomenon!