Chapter 3

NP Completeness

CS 573: Algorithms, Fall 2014
September 3, 2014

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

- ⋆ $s \in X$ if and only if
- ⋆ there is a string t:
  - (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) $X \in$ **NP**, and
(B) **(Hardness)** For any $Y \in$ **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** $=$ **NP**.

Proof: $\Rightarrow$ Suppose X can be solved in polynomial time
- (A) Let $Y \in$ **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.
- (C) Thus, every problem $Y \in$ **NP** is such that $Y \in P$; $NP \subseteq P$.
- (D) Since **P** $\subseteq$ **NP**, we have **P** $=$ **NP**.

$\Leftarrow$ Since **P** $=$ **NP**, and $X \in$ **NP**, we have a polynomial time algorithm for X.

3.1.0.4 NP-Hard Problems

(A) Formal definition:

> Definition 3.1.5. A problem X is said to be **NP-Hard** if
> (A) **(Hardness)** For any $Y \in$ **NP**, we have that $\mathsf{Y} \leq_P \mathsf{X}$.

(B) An **NP-Hard** problem need not be in **NP**!
(C) **Example:** Halting problem is **NP-Hard** (why?) but not **NP-Complete**.

3.1.0.5 Consequences of proving NP-Completeness

(A) If X is **NP-Complete**
  (A) Since we believe **P** $\neq$ **NP**,
  (B) and solving X implies **P** $=$ **NP**.
  X is **unlikely** to be efficiently solvable.
(B) At the very least, many smart people before you have failed to find an efficient algorithm for X.
(C) (This is proof by mob opinion — take with a grain of salt.)

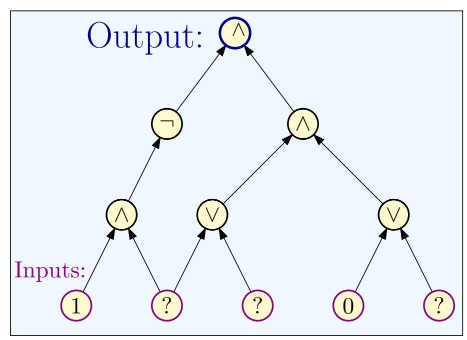
3.1.1 Preliminaries

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 **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

(A) Need to show that *every* **NP** problem X reduces to **CSAT**.
(B) What does it mean that $X \in$ **NP**?
(C) $X \in$ **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 ($s \notin X$) 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

(A) X is in **NP** means we have access to $p(), q(), C(\cdot,\cdot)$.
(B) What is $C(\cdot,\cdot)$? It is a program or equivalently a Turing Machine!
(C) How are $p()$ and $q()$ given?
  As numbers.
(D) Example: if 3 is given then $p(n) = n^3$.
(E) 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

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

3.1.2.6 Reducing X to CSAT

(A) **Q:** How do we reduce X to **CSAT**?
(B) Need algorithm **alg** that:
  (A) Input: s (and $\langle p, q, C \rangle$).
  (B) creates circuit G in poly-time in $|s|$ ($\langle p, q, C \rangle$ fixed).
  (C) G satisfiable $\iff$ $\exists$ proof t: $C(s,t)$ returns YES.

(C) **Simple but Big Idea:** Programs are the same as Circuits!
   (A) Convert $C(s,t)$ into a circuit G with t as unknown inputs (rest is known including s)
   (B) Known: $|t| \le p\big(|s|\big)$ 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) Formal definition:

Independent Set

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

(B) **Certificate:** Set $S \subseteq V$.
(C) **Certifier:** Check $|S| \ge k$ and no pair of vertices in S is connected by an edge.
(D) **Q:** Formally, why is **Independent Set** in **NP**?

3.1.3 Example: **Independent Set**

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

(A) 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 $\langle G, k \rangle$.
   (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: size of independent set.
(B) **Certificate**: $t = t_1 t_2 \ldots t_n$.
Interpretation: $t_i = 1$ if vertex i is in independent set.
... 0 otherwise.

3.1.3.2 Certifier for **Independent Set**

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

```
if (t_1 + t_2 + ... + t_n < k) then
    return NO
else
    for each (i, j) do
        if (t_i ∧ t_j ∧ y_{i,j}) then
            return NO

return YES
```

3.1.4 Example: Independent Set

3.1.4.1 A certifier circuit for Independent Set

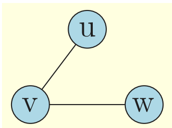
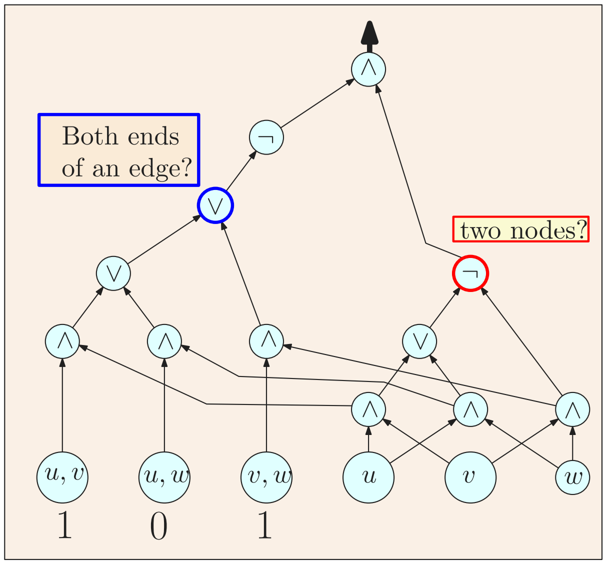


Figure 3.1: Graph G with $k = 2$



3.1.4.2 Programs, Turing Machines and Circuits

(A) **alg**: "program" that takes $f(|s|)$ steps on input string s.
(B) **Questions**: What computer is used?
What does *step* mean?
(C) "Real" computers difficult to reason with mathematically:
  (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
(D) 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)

3.1.4.3 Certifiers that at TMs

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

3.1.4.4 Simulation of Computation via Circuit

(A) M state at time ℓ: 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) Time 0: State of M = input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.

(C) Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
(D) Build circuit C_ℓ: Evaluates to YES
$\iff$ transition of M from time ℓ to time $\ell + 1$ valid.
(Circuit of size $O\Big(q(|s|)\Big)$.
(E) $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$.
Polynomial size!
(F) Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

3.1.4.5 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 $\mathcal{A}$, 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.