CS 573: Algorithms, Fall 2014

NP Completeness

Lecture 3

September 3, 2014

Part I

NP Completeness

Certifiers

Definition

An algorithm $\boldsymbol{C}(\cdot,\cdot)$ is a ***certifier*** for problem $\boldsymbol{X}$ if for every $s \in \boldsymbol{X}$ there is some string t such that $\boldsymbol{C}(s,t) =$ "yes", and conversely, if for some s and t, $\boldsymbol{C}(s,t) =$ "yes" then $s \in \boldsymbol{X}$.
The string t is called a certificate or proof for s.

Definition (Efficient Certifier.)

A certifier $\boldsymbol{C}$ is an ***efficient certifier*** for problem $\boldsymbol{X}$ if there is a polynomial $p(\cdot)$ such that for every string s, we have that

- $s \in \boldsymbol{X}$ if and only if
- there is a string t:
    1. $|t| \le p(|s|)$,
    2. $\boldsymbol{C}(s,t) =$ "yes",
    3. and $\boldsymbol{C}$ runs in polynomial time.

NP-Complete Problems

Definition

A problem $\boldsymbol{X}$ is said to be **NP-Complete** if

1. $\boldsymbol{X} \in$ **NP**, and
2. (Hardness) For any $\boldsymbol{Y} \in$ **NP**, $\mathbf{Y} \le_P \mathbf{X}$.

Solving NP-Complete Problems

Proposition

Suppose $\boldsymbol{X}$ is **NP-Complete**. *Then $\boldsymbol{X}$ can be solved in polynomial time if and only if* $\mathbf{P} = \mathbf{NP}$.

Proof.

$\Rightarrow$ Suppose $\boldsymbol{X}$ can be solved in polynomial time

0.1 Let $\boldsymbol{Y} \in \mathbf{NP}$. We know $\mathbf{Y} \leq_P \mathbf{X}$.

0.2 We showed that if $\mathbf{Y} \leq_P \mathbf{X}$ and $\boldsymbol{X}$ can be solved in polynomial time, then $\boldsymbol{Y}$ can be solved in polynomial time.

0.3 Thus, every problem $\boldsymbol{Y} \in \mathbf{NP}$ is such that $\boldsymbol{Y} \in \boldsymbol{P}$; $\boldsymbol{NP} \subseteq \boldsymbol{P}$.

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

$\Leftarrow$ Since $\mathbf{P} = \mathbf{NP}$, and $\boldsymbol{X} \in \mathbf{NP}$, we have a polynomial time algorithm for $\boldsymbol{X}$. □

NP-Hard Problems

1. Formal definition:

Definition

A problem $\boldsymbol{X}$ is said to be **NP-Hard** if

1.1 (Hardness) For any $\boldsymbol{Y} \in \mathbf{NP}$, we have that $\mathbf{Y} \leq_P \mathbf{X}$.

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

Consequences of proving NP-Completeness

1. If $\boldsymbol{X}$ is **NP-Complete**
   1.1 Since we believe $\mathbf{P} \neq \mathbf{NP}$,
   1.2 and solving $\boldsymbol{X}$ implies $\mathbf{P} = \mathbf{NP}$.

   $\boldsymbol{X}$ is unlikely to be efficiently solvable.
2. At the very least, many smart people before you have failed to find an efficient algorithm for $\boldsymbol{X}$.
3. (This is proof by mob opinion — take with a grain of salt.)

NP-Complete Problems

Question

Are there any problems that are **NP-Complete**?

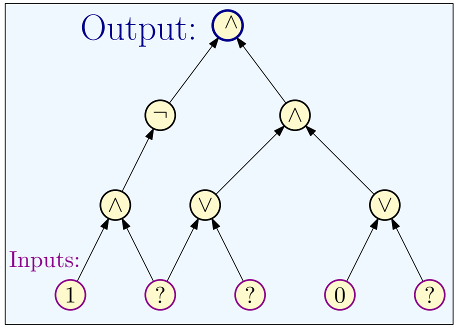
Answer

Yes! Many, many problems are **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 $\vee$, $\wedge$ or $\neg$.
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 $\boldsymbol{X}$ 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 $\boldsymbol{X}$ reduces to **CSAT**.
2. What does it mean that $\boldsymbol{X} \in$ **NP**?
3. $\boldsymbol{X} \in$ **NP** implies that there are polynomials $\boldsymbol{p()}$ and $\boldsymbol{q()}$ and certifier/verifier program $\boldsymbol{C}$ such that for every string $\boldsymbol{s}$ the following is true:
   - 3.1 If $\boldsymbol{s}$ is a YES instance ($\boldsymbol{s} \in \boldsymbol{X}$) then there is a *proof* $\boldsymbol{t}$ of length $\boldsymbol{p(|s|)}$ such that $\boldsymbol{C(s, t)}$ says YES.
   - 3.2 If $\boldsymbol{s}$ is a NO instance ($\boldsymbol{s} \not\in \boldsymbol{X}$) then for every string $\boldsymbol{t}$ of length at $\boldsymbol{p(|s|)}$, $\boldsymbol{C(s, t)}$ says NO.
   - 3.3 $\boldsymbol{C(s, t)}$ runs in time $\boldsymbol{q(|s| + |t|)}$ time (hence polynomial time).

Reducing X to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot,\cdot)$.
2. What is $C(\cdot,\cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if $\mathbf{3}$ is given then $p(n) = n^3$.
5. Thus an **NP** problem is essentially a three tuple $\langle p, q, C\rangle$ where C is either a program or a TM.

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 $p\big(|s|\big)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q\big(|s|\big)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **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 **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: $G = (V, E)$, k
   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**?

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.

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
```


Example: Independent Set

A certifier circuit for Independent Set

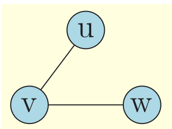
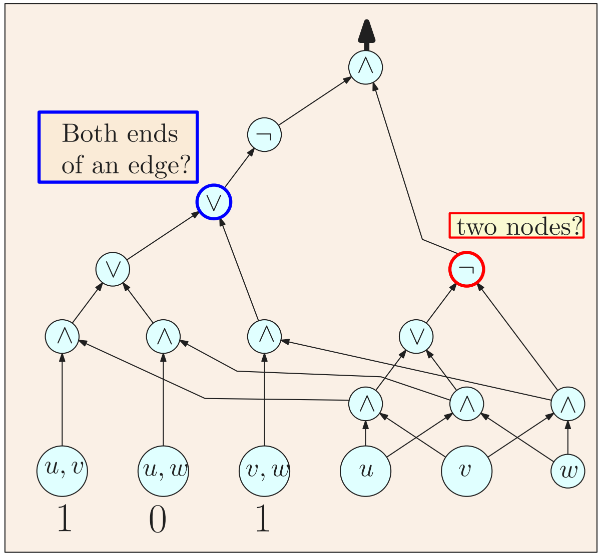


Figure:
Graph G
with $k = 2$



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:
   3.1 instruction set is too rich
   3.2 pointers and control flow jumps in one step
   3.3 assumption that pointer to code fits in one word
4. Turing Machines:
   4.1 simpler model of computation to reason with
   4.2 can simulate real computers with *polynomial* slow down
   4.3 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:
   2.1 $\exists$ proof t of length $\leq p(|s|)$
   2.2 M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      3.2.1 Uses at most $q(|s|)$ memory/tape cells.
      3.2.2 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.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of M = 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_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O\big(q(|s|)\big)$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$. Polynomial size!
6. Output of $\mathcal{C}$ 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 $\mathcal{A}$, 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.