More NP-completeness

CS255
Chris Pollett
Apr. 26, 2006.

Outline

- More on languages
- Polynomial-time verification
- NP-completeness and Reducibility
- Cook’s Theorem
- NP-complete problems

More on Languages

- We want to connect algorithms with languages.
- We say an algorithm A **accepts** a string x if A run on x outputs 1.
- If it outputs 0 it **rejects** the string.
- We say an algorithm A **accepts** a language L if the only strings it accepts are in L.
- We say a language is **decided** by A if A accepts the language and strings not in the language are rejected.
- A **complexity class** is a set of languages membership in which is determined by some **complexity measure**, for instance, runtime.
- For example, **P** is the complexity class of languages decided in polynomial time.
- It is also equivalently formulated as the class of languages accepted in polynomial time. (Just run polynomially many steps if it hasn't accepted yet, reject.)

Polynomial-Time Verification

- We now look at algorithms which can verify membership in languages.
- As an example…
- Call an undirected graph G **hamiltonian** if it contains a **hamiltonian cycle;** that is, a simple cycle which contain each vertex of G.
- Let HAM-CYCLE = $\{<G> \mid G$ is a hamiltonian graph$\}$
- How might one decide this problem? One could try each possible permutation of vertices. Let m be the number of vertices of the graph. Typically, $m = \Omega(\text{sqrt}(|<G>|))$. There are m! many permutations. So this algorithm would have exponential runtime.
- On the other hand, consider the language
  $H = \{<G, P> \mid P$ is a hamiltonian cycle in $G\}$.
  This language has a polynomial time decision algorithm. Further, the size of P is polynomial in the size of G, so we could rewrite HAM-CYCLE as:
  $\{<G> \mid \exists P, |P| \leq |G| \text{ and } <G, P> \in H\}$
- H can be viewed as verifying HAM-CYCLE in polynomial time.

The complexity class **NP**

- We are now ready to define the complexity class **NP**.
- We say a language L belongs to **NP** if there exists a two input polynomial-time algorithm A and a constant c such that
  $L = \{x \in \{0,1\}^* : \exists y, |y| = O(|x|^c) \text{ and } A(x,y) = 1\}$
- i.e., it is the class of languages that have polynomial time verification algorithms. So HAM-CYCLE ∈ **NP.**
- It is not hard to see **P⊆NP,** but it is unknown if **P=NP**.
- In fact, there is a million dollar prize to anyone who can solve this problem.
- Given a complexity class **C,** let **co-C** denote the class of languages whose complement is in **C**.
- One can see **P⊆NP∩co-NP**, but it is unknown if equality holds.

Polynomial-Time Reducibility

- There is some evidence to show that **P=NP** is unlikely.
- Further many problems have been shown to be in **NP**.
- So it is useful to be able to classify which **NP** problem are easy and which are hard.
- To do this, we say a language L_1 is **polynomial-time reducible** to language L_2, written $L_1 \leq_P L_2$ if there exists a polynomial time computable function f:$\{0,1\}^*$ --> $\{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1$ iff $f(x) \in L_2$.

Lemma. If L_1, L_2 are languages such that $L_1 \leq_P L_2$ and L_2 is in **P**, then L_1 is in **P**.

Proof. Let $A(y)$ decide L_2 in time $O(p(|y|))$. Let $f(x)$ be a $O(q(|x|))$-time reduction from L_1 to L_2. Here p and q are polynomials. Then $B(x)$ which first computes $f(x)$ then runs $A(f(x))$, runs in $O(p(q(|x|))$-time and decides L_1. So B run in polynomial time.

NP-completeness

- The p-time languages in **NP** are the easy languages.
- In contrast, a language L is called **NP-complete** if
  1. L is in **NP**, and
  2. $L' \leq_{\mathrm{P}} L$ for every L' in **NP**.
- A language which satisfies (2) but not necessarily (1) is called **NP-hard**.
- Let **NPC** denote the class of **NP**-complete languages.

Theorem. If any **NP**-complete language is in **P,** then **P=NP**.

Proof. This follows from the lemma on the last slide.

A first **NP**-complete problem

- Let CIRCUIT-SAT be the language:

{<*C*> | *C* is a AND, OR, NOT circuit computing a 0-1 function which on some truth assignment to its input variables outputs 1}

Theorem. CIRCUIT-SAT is in **NP**.

Proof. Consider the algorithm following algorithm *A*(<*C*>, <*a*>). First, *A* checks <*C*> is in the format of a circuit and <*a*> is in the format for an assignment; if not, it rejects. *A* then labels each of the inputs to <*C*> with their value according to their values in <*a*>. Then it loops over the combinational elements in <*C*>, until there is no change doing the following:

1. Check if the current element is not assigned a value but its children have been assigned a value.
2. Calculate the value of the node based on its gate type and its children.

By the *i*th iteration the nodes of depth *i* will have values. Each iteration involves less than quadratic work. So in $O((|\langle C\rangle|)^3)$ this algorithm labels the root of the circuit with its output value on this assignment. Finally, CIRCUIT-SAT is the language $\{\langle C\rangle \in \{0,1\}^* : \exists \langle a\rangle, |\langle a\rangle| \leq |C|$ and $A(\langle C\rangle,\langle a\rangle) = 1\}$.

Cook's Theorem

Theorem. CIRCUIT-SAT is **NP**-hard.

Proof. Let L be a language in **NP**, let $A(x,y)$ verify the language in time $O(|x|^c)$. The algorithm A runs on some kind of computational hardware. If that hardware is in a given configuration c_i then its control determines in the next time step what its next configuration c_{i+1}. We assume that this mapping can be computed by some AND, OR, NOT circuit M implementing the computer hardware. Using this circuit M. We build an AND, OR, NOT circuit $<C(y)>$ which is split into main layers which have the properties.:

1. The output of C at main layer 1 codes, c_0 , a configuration of M at the start of the computation of $A(x,y)$. Here the values of x are hard-coded based on the instance x which we are trying to check is in L. y is not hard-coded and boolean variables are used to represent it.
2. For each i, the output of C at main layer $i + 1$, corresponds to the configuration obtained from main layer i by computing according to M.
3. The output of C is the value extracted from the final configuration of A *after* $O(|x|^c)$ steps.

Since there are polynomially many main layers each separated by polynomial sized circuits, this whole circuit will be polynomial size. If there is some setting of the boolean variables for y which makes the circuit true, then $A(x,y)$ holds and x will be in L as desired.

NP-completeness Proofs

- In general, most NP-completeness proof will make use of the following lemma:

Lemma. If some **NP**-complete language reduces to a language *L*, then *L* is **NP**-hard. If *L* is further in **NP** then *L* will be **NP**-complete.

Proof. Just compose the reductions.

Some NP-complete Problems

- Let SAT={<*F*>| <*F*> is a satisfiable boolean formula}
- Let 3SAT={<*F*>| <*F*> is a satisfiable CNF formula where each clause has at most three literal}.

Theorem. Both SAT and 3SAT are NP-complete.