CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 22

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we will look at Reductions and NP-completeness.

2 Polynomial Reductions:

Consider two decision problems A and B. We say A is polynomially reducible to B if there exists a polynomial time algorithm that converts each input a to A and another input b to B such that the answer to input a for problem A is YES if and only if the answer to input b for problem B is YES. Here we assume the size of b is also polynomial in the size of a. For example, the reductions from undirected graphs to directed ones for shortest paths.

Theorem 1 *If A is polynomially reducible to B and there is a poly time algorithm for B, then there is a poly time algorithm for A as well.*

Proof: First reduce the input of A to B and then solve it. ■

Theorem 2 *If A is polynomially reducible to B and B is polynomially reducible to C, then A is polynomially reducible to C (transitivity).*

Proof: Just compose the two poly time algorithms in the reductions. ■

3 Two Important Complexity Classes

Now we define two classes, which not only contain numerous important problems (all equivalent to one another) that are not known to be in P, but also contain the hardest problems in NP.

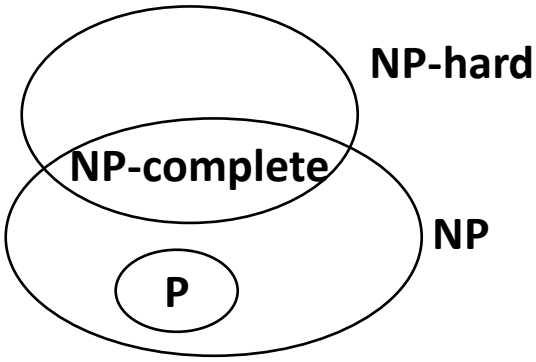

Figure 1: If $\mathsf{P} \neq \mathsf{NP}$

Definition 1 *A problem* X *is called* NP***-hard*** *if every problem in* NP *is polynomially reducible to* X.

Definition 2 *A problem X is called* ***NP-complete*** *if* $\mathsf{X} \in \mathsf{NP}$ *and* X *is* NP*-hard.*

Note that the definition of NP-hardness implies that of any NP-hard problem is ever proved to belong to P, then that proof would imply that $\mathsf{P} = \mathsf{NP}$ (by Theorem 1) and you win the 1 million prize. Cook in 1971 showed that there is at least one NP-complete problem called Satisfiability (SAT). Now our job is much easier. Why?

Theorem 3 X *is* NP*-complete if* $\mathsf{X} \in \mathsf{NP}$ *and for some problem* Y *that is NP-complete, Y is polynomially reducible to X.*

Proof: By definition of NP-completeness we know that every problem in NP is polynomially reducible to Y. But since Y is polynomially reducible to X and reducibility is transitive (Theorem 2), every problem is polynomially reducible to X as well. ■

4 Satisfiability (SAT)

Let S be a Boolean expression in Conjunctive Normal Form (CNF). That is, S is a AND (product) of several ORs (sum), e.g., $\mathsf{S} = (\mathsf{x}+\mathsf{y}+\overline{\mathsf{z}})(\overline{\mathsf{x}}+\mathsf{y}+\mathsf{z})(\overline{\mathsf{x}}+\overline{\mathsf{y}}+\overline{\mathsf{z}})$. Each variable is either 0 (false) or 1 (true). One can show that any Boolean expression can be transformed into CNF.

Definition 3 *A Boolean expression is said to be* ***satisfiable*** *if there exists an assignment of 0s and 1s to its variables such that the value of the expression is 1.*

The **SAT** problem is to determine whether a given expression is satisfiable (without necessarily finding a satisfying assignment). For example for the S above, $x = 1, y = 1$ and $z = 0$ makes the expression 1. We call an assignment of 0s and 1s to the variables of a Boolean expression a **truth assignment**. Note that using a backtracking algorithm, we can solve SAT in $O(2^n n)$ time.

4.1 Cook's Theorem

Theorem 4 *SAT is NP-complete.*

Note that $SAT \in NP$ is easy to see. The proof of Cook's theorem however is complicated and uses modeling of a non-deterministic computer with SAT. Note that a few other problems such as graph coloring and bin packing that you have seen so far are NP-complete as well (that is the reason why we design approximation algorithms for these problems). Now let us see more examples.

5 3-SAT is NP-complete

3-SAT Problem:
Given a Boolean expression in CNF such that each clause contains exactly three variables, determine whether it is satisfiable.

The problem might seem easier than SAT since there is an additional requirement of exactly three variables per clause. Now we see a proof that 3-SAT is NP-complete.

Theorem 5 *3-SAT is NP-complete*

Proof: First the problem is clearly in NP since given a certificate of a truth assignment we can verify it easily in poly time.

Let $C = (x_1 + x_2 + \ldots + x_k)$ be an arbitrary clause of the input to SAT with k variables. We write each variable in its "positive" form (i.e. we do not use $\overline{x_i}$) only for convenience of the problem. The following four cases need to be considered:

- If $k = 3$, we are done.
- If $k = 2$ and $C = x_1 + x_2$, then we replace it with $C' = (x_1 + x_2 + z)(x_1 + x_2 + \overline{z})$.
- If $k = 1$ and $C = x_1$ then $C' = (x_1 + y + z)(x_1 + \overline{y} + z)(x_1 + y + \overline{z})(x_1 + \overline{y} + \overline{z})$ where both y and z are new variables. Note that C is satisfiable if and only if C' is satisfiable.
- If $k \geq 4$ then we replace $C = (x_1 + x_2 + \ldots + x_k)$ by $C' = C_1 C_3 C_4 \ldots C_{k-2} C_k$ where $C_1 = (x_1 + x_2 + y_1)$ and $C_k = (\overline{y_{k-3}} + x_{k-1} + x_k)$. For every $3 \leq i \leq k-2$, we set $C_i = (\overline{y_{i-2}} + x_i + y_{i-1})$. Now if C is satisfiable, then

one of the $x_i's$ is set to 1. Then we can set the values of the $y_j's$ accordingly so that all clauses in C' evaluate to 1 as well. If $x_1 = 1$ or $x_2 = 1$, then set all y-variables to 0. If $x_{k-1} = 1$ or $x_k = 1$ then set all y-variables to 1. If $x_i = 1$ for some $3 \leq i \leq k-2$, then set $y_1, y_2, \ldots, y_{i-2}$ to 1 and all other y-variables to 0.

■

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*