Com S 311 Section B
Introduction to the Design and Analysis of Algorithms

Lecture One for Week 12

Xiaoqiu (See-ow-chew) Huang

Iowa State University

April 13, 2021

NP-Completeness and Reducibility

The most compelling reason why computer scientists believe that $P \neq NP$ is that there exists a class of NP-complete problems.

This class guarantees that if *any* NP-complete problem can be solved in polynomial time, then *every* problem in NP has a polynomial-time solution. So $P = NP$.

No polynomial-time algorithm has ever been found for any NP-complete problem.

Below we use polynomial-time reducibility to fommally define the NP-complete languages and justify that a language called CIRCUIT-SAT is NP-complete.

Reducibility

Intuitively, a problem Q can be reduced to another problem E if any instance of Q can be rephrased as an instance of E such that the solution to the E instance corresponds to the solution to the Q instance.

Formally, a language L_1 is **polynomial-time reducible** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function f from $\{0,1\}^*$ to $\{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L_1$ if and only if $f(x) \in L_2$.

The function f is called the **reduction function**, and a polynomial-time algorithm F computing f is called a **reduction algorithm**.

Using Polynomial-Time Reductions

Lemma 34.3

Let L_1 and L_2 be languages over $\{0, 1\}$ such that $L_1 \leq_P L_2$. Then if L_2 is in P, then L_1 is also in P.

Proof

Assume that L_2 is decided by a polynomial-time algorithm A_2 and that the reduction function f is computed by a polynomial-time algorithm F.

Then L_1 is decided by a polynomial-time algorithm A_1, which transforms a input string $x \in \{0, 1\}^*$ into $f(x)$ and runs A_2 on $f(x)$, reporting the output from A_2.

Because $x \in L_1$ if and only if $f(x) \in L_2$, the algorithm A_1 is correct.

NP-Completeness

We use polynomial-time reductions to define the set of NP-complete languages, which are the hardest problems in NP.

A language $L \subseteq \{0,1\}^*$ is **NP-complete** if
1. $L \in \mathrm{NP}$, and
2. $L' \leq_P L$ for every $L' \in \mathrm{NP}$.

If a language L satisfies property 2, but not necesarily property 1, then L is **NP-hard**. The class NPC is the class of NP-complete languages.

NP-Completeness in Deciding If P = NP

Theorem 34.4
If any NP-complete language is in P, then P = NP. Equivalently, if any language in NP is not in P, then no NP-complete language is in P.

Proof
Assume that $L \in \mathrm{NPC}$ is in P. Then for any $L' \in \mathrm{NP}$, we have $L' \leq_P L$ by property 2. Thus, by Lemma 34.3, we also have that $L' \in \mathrm{P}$, implying $\mathrm{NP} \subseteq \mathrm{P}$. Since $\mathrm{P} \subseteq \mathrm{NP}$, we have P = NP.

The second statement is the contrapositive of the first one.

Circuit Satisfiability

We show that the circuit satisfiability problem is NP-complete.

A **boolean combinational circuit** consists of one or more boolean combinational elements (**logic gates** such as the **NOT gate**, the **AND gate** and the **OR gate**) interconnected by **wires**, where a wire allows the output value of one element to be used as an input value of another.

The number of element inputs fed by a wire is the **fan-out** of the wire.

If no element output is connected to a wire, the wire is a **circuit input**.

If no element input is connected to a wire, the wire is a **circuit output**, where the number of circuit outputs is limited 1 for the circuit-satisfiability problem.

Circuit Satisfiability

A **truth assignment** for a boolean combinational circuit is a set of boolean input values.

A one-output boolean combinational circuit is **satisfiable** if there exists a **satisfying truth assignment** to cause the circuit to output 1.

The **circuit-satisfiability problem** is to determine whether a given boolean combinational circuit composed of AND, OR, and NOT gates is satisfiable.

We use an encoding to map any given boolean circuit C to a binary string $< C >$ whose length is polynomial in the size of the circuit.

The circuit-satisfiability problem is defined as a formal language:
CIRCUIT-SAT $= \{ < C >: C \text{ is a satisfiable boolean circuit } \}$.

CIRCUIT-SAT is in NP

Lemma 34.5
The circuit-satisfiability problem is in the class NP.

Proof
We design a two-input, polynomial-time algorithm A for verifying CIRCUIT-SAT.

One of the inputs to A is a standard encoding of a boolean combinational circuit C, and the other is a certificate corresponding to an assignment of boolean values to the wires of C.

For each logic gate in the circuit, the algorithm A checks that the value on the output wire is correctly computed as a function of the values on the input wires. If the output of the entire circuit is 1, A outputs 1. Otherwise, it outputs 0.

Proof

When a satisfiable circuit C is input to algorithm A, there exists a certificate whose length is polynomial in the size of C and that causes A to output 1.

when an unsatisfiable circuit is input, no certificate can cause A to output 1.

Algorithm A runs in polynomial time, so CIRCUIT-SAT is in NP.

CIRCUIT-SAT Is NP-Complete

Next we show that every language in NP is polynomial-time reducible to CIRCUIT-SAT.

We give an informal proof based on some understanding of the workings of computer hardware.

A computer program is stored in the computer memory as a sequence of instructions.

A typical instruction encodes an operation to be performed, addresses of operands in memory, and an address where the result is to be stored.

A special memory location, called the **program counter**, keeps track of which instruction is to be executed next.

CIRCUIT-SAT Is NP-Complete

Any particular state of computer memory (including the program itself, the program counter, working storage) is called a **configuration**.

The excution of an instruction is viewed as mapping one configuration to another.

The computer hardware that performs this mapping can be implemented as a boolean combinational circuit, which is denoted by M in the following lemma.

Lemma 34.6
CIRCUIT-SAT Is NP-hard.

Proof

Consider any language L in NP. We describe a polynomial-time algorithm F computing a reduction function f that maps every binary string x to a circuit $C = f(x)$ such that $x \in L$ if and only if $C \in \mathrm{CIRCUIT\text{-}SAT}$.

Let L be verfified by an algorithm A in worst-case polynomial time $T(n)$ in length-n input strings.

Let k be a constant such that $T(n) = O(n^k)$ and the length of the certificate is $O(n^k)$.

The computation of algorithm A is represented as a sequence of configurations: $c_0, c_1, c_2, ..., c_{T(n)}$. Each configuration consits of the program for A, the program counter and auxiliary machine state, the input x, the certificate y, and working storage.

Proof

The combinational circuit M, which implements the computer hardware, maps each configuration c_i to the next one c_{i+1}.

The reduction algorithm F constructs a single combinational circuit C that computes all configurations produced by a given initial configuration:

$$c_0 \rightarrow M \rightarrow c_1 \rightarrow M \rightarrow c_2 \rightarrow \ldots \rightarrow M \rightarrow c_{T(n)}.$$

The output of the ith circuit, which produces configuration c_i, feeds directly into the input of the $(i+1)$st circuit.

The configurations, rather than being stored in the computer's memory, simply reside as values on the wires connecting copies of M.

The output for the entire combinational circuit C is the $0/1$ output of algorithm A in the configuration $c_{T(n)}$.

Proof

We can show that C is satisfiable if and only if there exists a certificate y such that $A(x, y) = 1$.

We can also show that the reduction algorithm F runs in polynomial time.