Lecture 7: NP-completeness

Anup Rao

April 16, 2024

Polynomial time Reductions

One of the central questions in complexity theory is whether or not $\mathbf{P} = \mathbf{NP}$. Although we don't know the answer to this question, we can prove a lot about the class **NP**, via the concept of polynomial time reductions:

Definition 1. *A function f is polynomial time reducible to a function g if there is a polynomial time computable function h such that $f(x) = g(h(x))$. We write $f \leq_P g$.*

Note that the above definition is not the only one that makes sense. In general it makes sense to allow our reductions to make multiple calls to the problem being reduced to. However, we will be able to prove many of our results using the stronger notion above, so that is what we shall use.

Definition 2. *We say f is* **NP***-hard if $g \leq_P f$ for every $g \in \mathbf{NP}$. We say f is* **NP***-complete if f is* **NP***-hard and $f \in \mathbf{NP}$.*

Theorem 3. *Here are some easy facts that one can prove about reductions:*

- *If $f \leq_P g$ and $g \leq_P h$, then $f \leq_P h$.*
- *If f is* **NP***-hard and $f \in \mathbf{P}$, then $P = NP$.*
- *If f is* **NP***-complete, then $\mathbf{P} = \mathbf{NP}$ if and only if $f \in P$.*

NP*-complete problems*

The above definitions make sense because we do know of examples of **NP**-complete problems.

Circuit-Sat

Definition 4. *$CircuitSat : \{0,1\}^* \to \{0,1\}$ is the function that views its input as a circuit C and outputs 1 iff $\exists x$ such that $C(x) = 1$.*

I have claimed in class that circuits can simulate Turing Machines. Here is what you can actually prove in this regard:

Theorem 5. *If a function $f : \{0,1\}^* \to \{0,1\}$ can be computed in time $t(n)$ by a Turing machine, then for every n there is a circuit of size $O(t(n) \log t(n))$ that computes f restricted to the inputs of size n.*

Although we did not prove this theorem in class, we sketched how you could find a circuit of size $O(t(n)^2)$ that computes f. The idea was to add a layer of gates that maintains the entire state of the Turing machine—contents of all tapes, pointers, and the line of code being executed. Then we add a new layer that computes this configuration after one execution step of the Turing machine, using the earlier configuration as input. A single configuration can be written down using $O(t(n))$ gates since we only need to write down the values of the tapes up to $O(t(n))$ coordinates. The new configuration can be computed from the old one with $O(t(n))$ gates as well. After repeating this $O(t(n))$ times, we obtain the final configuration of the Turing machine, which must include the value of $f(x)$.

Theorem 6. *CircuitSat is* **NP***-complete.*

Proof It is clear that *CircuitSat* is in **NP**. Next we show that for every $f \in \mathbf{NP}$, $f \leq_P$ *CircuitSat*. Let V be a verifier for f. Then to compute $f(x)$, the reduction will build the circuit $C_x(w)$ that computes $V(x, w)$, where here w are the input variables to the circuit and x is the input. Since $f(x) = 1$ if and only if there exists w such that $C_x(w) = 1$, we can determine the value of f by computing $CircuitSat(C_x)$. ■

3SAT

A *boolean formula* is an expression of the form

$$(x_1 \wedge \neg x_2) \vee (x_7 \wedge \neg(x_6 \vee \neg x_2)).$$

Formally: it is a circuit where the only allowed gates are $\vee, \wedge, \neg$, and every gate has fan-out at most 1. Input gates are allowed to repeat. As usual, size of the gates is number of gates, and the fan-in is allowed to be at most 2. The formula is said to be in conjunctive normal form (CNF) if it is an AND of OR's. Similarly, it is said to be in disjunctive normal form (DNF) if it is an OR of ANDS. For example

$$(x_1 \vee \neg x_2) \wedge (\neg x_7 \vee x_9 \vee \neg x_1)$$

is a CNF.

We have the following lemma:

Lemma 7. *Every function $f : \{0,1\}^\ell \to \{0,1\}$ can be computed by a CNF (resp. DNF) of size $\ell 2^\ell$.*

Proof For each input z such that $f(z) = 0$, we add the literal x_i to the clause if $z_i = 0$ and $\neg z_i$ otherwise. So for example, if $f(0,1,0) =$

0, we add the clause $(x_1 \vee \neg x_2 \vee x_3)$. Then note that each clause is 0 on exactly one input, and all inputs x for which $f(x) = 0$ make some clause 0. Every other input evaluates to 1. So, the CNF computes f. The resulting formula is of size $\ell 2^\ell$. The case of DNF's is symmetric. ■

We define $\mathsf{SAT} : \{0,1\}^* \to \{0,1\}$ to be the function that takes as input a boolean formula F, and outputs 1 if and only if there is a an x such that $F(x) = 1$. A 3-CNF formula is a CNF where every clause has at most 3 variables. For example:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \neg x_1) \wedge \cdots .$$

$\mathsf{3SAT} : \{0,1\}^* \to \{0,1\}$ is the function that takes as input 3-CNF and outputs 1 if and only if the formula is satisfiable. Next we show that even this function is **NP**-complete

Is the same true for 2SAT? We do not know. There are polynomial time algorithms for 2SAT, so if you found a reduction to 2SAT, you would prove $\mathbf{P} = \mathbf{NP}$. The algorithm works by viewing every clause $(x \vee y)$ as an implication $\neg x \Rightarrow y$ as well as the implication $\neg y \Rightarrow x$. This defines a directed graph where all the vertices correspond to variables and their negations, and the edges correspond to implications. You can show that the formula is satisfiable if and only if there is no path that leads from a variable to its negation.

Theorem 8. 3SAT *is* **NP***-complete.*

Proof $\mathsf{3SAT} \in \mathbf{NP}$ is easy enough to check. The witness is a satisfying assignment to the formula. The verifier simply evaluates the formula on the given witness, and outputs the results of the evaluation.

Since we have already shown that $\mathsf{CKT-SAT}$ is **NP**-hard, it will be enough to show that $\mathsf{CKT-SAT} \leq_P \mathsf{SAT}$.

Given a circuit, we shall output a CNF formula that is satisfiable if and only if the circuit accepts some input. Introduce a new variable y_g for each internal gate g of the circuit. If the internal gate g has inputs h, q, let F_g be the CNF formula on variables y_g, y_h, y_q that is 1 if and only if $y_g = g(y_q, y_h)$. By Lemma 7, this formula is a 3-CNF of constant size. If the output gate is v, the final formula is

$$y_v \wedge \bigwedge_g F_g,$$

which is satisfied if and only if the circuit has a satisfying assignment.

Every clause of this formula has at most 3 variables. To make sure it has *exactly* 3 variables, we replace each clause with less than 3 variables with a 3-CNF that by adding dummy variables. For example, we can replace y_v by a 3-CNF on the variables y_v, z_1, z_2 that computes the same function as y_v:

$$(y_v \vee z_1 \vee z_2) \wedge (y_v \vee \neg z_1 \vee z_2) \wedge (y_v \vee \neg z_1 \vee \neg z_2) \wedge (y_v \vee z_1 \vee \neg z_2).$$

■