Lecture 4

August 18, 2014

1 NP-Completeness

Definition 1.1. Polynomial-time reduction: A language $L \subset \{0,1\}^*$ is said to be polynomial time reducible to another language L' if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $x \in L \iff f(x) \in L'$.

If the language L is reducible to L' then for every given string x we can simply check if $f(x) \in L'$ and conclude if $x \in L$. Time taken for this is sum of time taken to solve the problem in L' and a polynomial time for computing $f(x)$.

Definition 1.2. NP-hardness: A language L_H is said to be NP-hard if for every language $L \in NP$, $L \leq_p L_H$.

If an NP-hard problem can be solved in a certain amount of time, then every problem in NP can be solved in the same amount of time with just an additional polynomial factor.

Definition 1.3. NP-completeness: A language L_C is said to be NP-complete if $L_C \in$ NP and L_C is NP-hard.

Hence NP-complete problems are the hardest among all problems in NP. Existence of such problems is not guaranteed by the definition. Here is an example to show they exist.

Example 1.4. Define $L_c := \{< \alpha, x, 1^n, 1^t >: \exists u \in \{0,1\}^n$ such that M_α accepts $< x, u >$ in exactly t steps$\}$.

Proof. Claim 1: $L_c \in NP$
Consider a Turing Machine M_v. It accepts as input $\{< \alpha, x, 1^n, 1^t >, u\}$. On accepting input, it first checks if $|u| = n$. It then simulates the turing machine $M_\alpha(x, u)$ for atmost t steps. If the simulation outputs 1, it means that the specific input is a part of the language L_c.
Thus u the certificate for input x on M_α, also acts as a certificate for the input $< \alpha, x, 1^n, 1^t >$ to decide if it is in L_c. Also the first part of M_v takes n steps to

complete and the second part of M_v runs for atmost t steps. Thus the running time of M_v is $\Omega(n+t)$. Hence M_v is a deterministic polynomial-time verifier for L_c. Hence $L_c \in NP$.
Claim 2: $L_c \in$ NP-Hard
We need to show that every language $L \in NP$ is polynomial-time reducible to L_c.
Since $L \in NP$, we know that it has a polynomial-time verifier M_L. Let M_L have the binary representation α_L. Also there exists a certificate U_x for every input x. Further we know that $M_L(x, u_x)$ runs in time polynomial of the size of the input. Let the time taken to run be $p(|x|)$. Hence the size of u_x must also be polynomial in $|x|$. Let it be $q(|x|)$.
Now, consider the reduction

$$f : x \rightarrow < M_L, x, 1^{q(|x|)}, 1^{p(|x|+q(|x|))} >$$

By the construction above, $x \in L \iff < M_L, x, 1^{q(|x|)}, 1^{p(|x|+q(|x|))} > \in L_c$. Hence every language in NP is poly-time reducible to L_c. So $L_c \in$ NP-Hard. Thus L_c is NP-complete. □

The above example is a specifically constructed one and is not an organic example. We shall see some more natural examples in the next section.

2 Satisfiability problem

Definition 2.1. The language SAT is defined as $SAT = \{\phi : \exists x \in \{0,1\}^* \text{ such that } \phi(x) = 1\}$.

Claim 1. SAT $\in$ NP

Claim 2. SAT $\in$ NP-complete

Definition 2.2. The language CNF-SAT is defined as CNF-SAT= $\{\phi : \phi$ is represented in the conjunctive normal form and ϕ is satisfiable$\}$.

Note: We say that ϕ is in conjunctive normal form if:
$\phi = c_1 \cap c_2 \cap \ldots \cap c_m$
$c_i = x_{i1} \bigvee x_{i2} \bigvee \ldots \bigvee x_{ik}$

Definition 2.3. The language 3-SAT is defined as $CNF - SAT = \{\phi : \phi$ is represented as a 3-CNF and ϕ is satisfiable$\}$.

Claim 3. 3-SAT is NP-complete

Proof. Assume CNF-SAT is NP-Complete.
A certificate for 3-SAT, or any satisfiability problem in general, is the satisfying assignment to the literals. Given an assignment, it can be checked in time proportional to the size of the boolean formula if it satisfies the formula or not. Hence 3-SAT is in NP.

Now we reduce the CNF-SAT problem to 3-SAT problem.
Consider any boolean formula in the conjunctive normal form. So $\phi = c_1 \cap c_2 \cap \ldots \cap c_m$ and $c_i = x_{i1} \bigvee x_{i2} \bigvee \ldots \bigvee x_{ik}$.
Let $c_{i1} = x_{i1} \bigvee x_{i2} \bigvee \ldots \bigvee x_{i(k-2)} \bigvee z_i$ and
$c_{i2} = x_{i(k-1)} \bigvee x_{ik} \bigvee \bar{z_i}$
Now if c_i has a solution, then one of the x_{ij} must be 1. Setting the z_i term in the same clause to 0, both the clauses will be satisfied. Also if the clause is not satisfiable, then all the x_{ij} must be 0 and z_i terms prevent both formulas from being satisfied simultaneously. Hence c_i is satisfiable if and only if $c_{i1} \bigvee c_{i2}$ is satisfiable.
We can proceed inductively to reduce the clause to a conjunction of clauses with at most 3 literals. Proceeding similarly the entire formula ϕ can be written in the 3-CNF form. In each of the new clauses, by construction, at least one of the original n literals exists. Thus the new number of clauses is at most $n \times max(k_i)$. The new formula can be computed in as much time as well.
Hence the above reduction is a deterministic poly-time reduction of CNF-SAT to 3-SAT. Hence 3-SAT is NP-Complete.

□

Note: It is interesting to note that by the above construction we cannot reduce the formula to 2-CNF form. The above iterative step will then remove one variable and create a new variable. The process will hence not terminate.
It would also be a good time to find polynomial time algorithm to solve 2-SAT.

Theorem 2.4 (Cook-Levin)**.** CNF-SAT is NP-Complete

Proof. As was stated before, the satisfying assignment serves a certificate for any given input and hence CNF-SAT is in NP.
Let $L \in NP$. We need to show that there exists a polynomial-time reduction from L to CNF-SAT.
By the definition of NP, there is a polynomial time verifier M_L and a certificate of polynomial length, $q(|x|)$ such that $x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \& M_L(< x, u >) = 1$. We also know that M_L runs in polynomial time. Let it run in time $p(|x|)$.
Consider the simulation of the turing machine M_L. Let it have a bit representation as:

x_1	x_2	$\ldots$	$x_{\|x\|}$	u_1	u_2	$\ldots$	$u_{q(\|x\|)}$	$\ldots$	$p(\|x\|)$	$\ldots$

Since the running time of M_L is $p(|x|)$, the head never goes beyond the cell $p(|x|)$ and we need not consider the tape beyond it.
Now consider expanding each cell in the tape above to contain three parts:

- First part contains the same bit as before
- Second part stores a 1 if the head of the machine is at the current state and 0 otherwise
- Third part stores the state of the machine if the head is in that state or a redundant state otherwise.

The bit-size of each cell still remains a constant. Now we look at the screenshot of the tape after each computation made by the machine. The machine can make at most $p(|x|)$ computations. So we look at all the $p(|x|)$ versions of the tape. We initialize the first tape with the values of x_i and u_i in their respective places and for every other bit in the first tape and every bit on every other tape we create new boolean variables z_i. Also, we assume that there is a certain output bit on the $p(|x|)$th screenshot of the tape that contains 1 if the string is accepted and zero otherwise.

x_1	z_{12}	z_{13}	$\ldots$	u_1	z_{1j}	$z_{1(j+1)}$	$\ldots$	$z_{1p(\|x\|)}$
z_{21}	z_{22}	z_{23}	$\ldots$	$z_{2(j-1)}$	z_{2j}	$z_{1(j+1)}$	$\ldots$	$z_{2p(\|x\|)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
o/p	$z_{p(\|x\|)2}$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$z_{p(\|x\|)p(\|x\|)}$

By the property of the turing machines, the head can move only one cell to the right or left in each step. The variable in the first compartment can be altered only if the head of the TM is on that cell and depends only on the state the TM is in. The head can move to a cell only if it is on an adjacent cell. The state variable is redundant unless the head is on the cell and depends on the state of the machine and the input being read. Hence the values of each cell in the tape depends only on the values in the the cell itself and its two adjacent cells in the previous time-step.
To obtain a 1 in the output cell, we need a certain configuration in the corresponding three cells. This relation can be easily expressed in the DNF form which can be converted to the CNF form. Similarly for all the cells, the value is constrained only by the three corresponding cells and these requirements can all

be represented as a boolean formula in the CNF form since they are all boolean variables. These constraints are sufficient to ensure the machine is not violating any of its properties.
If the machine does not accept an input $< x, u >$, then there cannot be a satisfying assignment of the boolean variables and if it accepts the input, then the satisfying assignment is the contents of the tape as the turing machine executes.
The number of clauses in the CNF representation of the machine is less than the number of boolean variables which is bounded by $O(cp(|x|)^2)$ which is polynomial as well.
Hence the construction provides a polynomial time reduction from the language L to CNF-SAT. Thus CNF-SAT is NP-Complete. □

Note: By the above construction the time taken for the reduction is $O(p^2(|x|))$. There are more efficient reductions that take only $O(p(|x|) \log p(|x|)$ time.

Notes scribed by:
Sabareesh R
ugsabareesh@ug.iisc.in