Lecture 3

In this lecture we will wrap up our discussion of **NP**-completeness by studying a new **NP**-complete problem, Independent Set, and showing how 3SAT reduces to it. Then we will consider non-deterministic models of computation and their relation to **NP**. Finally, we will introduce some space-bounded complexity classes and examine their relation to the time-bounded classes.

1 Independent Set is NP-Complete

We have already seen the proof that SAT and 3SAT are **NP**-complete. Now we will look at another **NP**-complete problem, Independent Set (IS). We can state problem IS as follows: given a graph $G = (V, E)$ and an ingeger $k > 0$, is there an independent set in G of size k? (A set of vertices $I \subseteq V$ is *independent* if no two vertices $u, v \in I$ are connected by an edge $(u, v) \in E$.)

There is an optimization version of this problem, which is to find the largest possible independent set of a given graph, but we are simply interested in the decision problem IS given above. We will show that 3SAT reduces to IS, hence IS is **NP**-complete.

THEOREM 1
3SAT $\leq$ IS.

PROOF: Say we have a problem in 3SAT which is a boolean expression ϕ with n variables $x_1, x_2, \ldots, x_n$, and m clauses. Construct a graph G with $3m$ vertices, one for each literal (an occurance of a variable in a clause). Add edges to connect vertices corresponding to literals belonging to the same clause, so that for each clause we have a triangle of vertices connecting its literal. Finally, add edges to connect all pairs of vertices which are negations of each other.

We claim that 3SAT for ϕ corresponds to IS for (G, m). First, suppose there *is* an independent set I of size m. Then it must have exactly one vertex from each triangle, since there are m triangles, and having two vertices from the same triangle would violate independence. Let $x_i = true$ if at least one x_i vertex is in I, $x_i = false$ if at least one $\overline{x_i}$ is in I, and $x_i = true$ if neither condition holds. (The first two conditions cannot both occur for the same x_i, since we constructed edges between all contradictory pairs of literals.) By construction, this satisfies ϕ, because for each clause we have made at least one of the three assignments that will satisfy it.

Conversely, if ϕ is satisfiable, pick a satisfying assignment of variables, and define I to contain, for each triangle, one arbitrarily chosen literal satisfied by the assignment. (At least one must exist, or else ϕ would not be satisfied!) This I is independent since edges only connect literals within the same clause (of which only one is in I) and literals which are negations of each other (of which only one would be consistent with our satisfying variable assignment).

This proves that ϕ can be satisfied if and only if G has an independent set of size m. G can obviously be constructed from ϕ in polynomial time. Therefore 3SAT $\leq$ IS. □

2 Nondeterministic Turing Machines

A Turing machine is usually described by a set of states Q, an alphabet Σ, and a state transition function

$$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, S, R\}$$

which, given the state of the machine and the symbol at the current position of the tape, tells the machine a new state to go to, a new symbol to write, and a direction to move in (left, right, or stay in the same place). A *nondeterministic* Turing machine instead has a function

$$\delta : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \times \{L,S,R\}}$$

which prescribes multiple possible things for the machine to do next. We say that a nondeterministic machine M solves a problem L if for every $x \in L$, there is a possible computation of $M(x)$ leading to acceptance, and for every $x \notin L$, all computations of $M(x)$ reject.

THEOREM 2
NP *is the set of decision problems solvable in polynomial time by a nondeterministic Turing machine.*

PROOF: Consider $L \in \mathbf{NP}$. We know there is an algorithm $V_L(\cdot,\cdot)$ running in polynomial time $T_L(\cdot)$ and a polynomial $p(\cdot)$ such that

$$x \in L \Leftrightarrow \exists y, |y| \leq p(|x|) \text{ and } V_L(x, y) \text{ accepts}$$

We can construct a machine M that uses nondeterminism to write all possible strings $y \in \{0,1\}^{p(|x|)}$ to its tape and then simulate $V_L(x, y)$, essentially testing all possible solutions in parallel. Thus this nondeterministic machine solves L in polynomial time.

Conversely, if L is solved in polynomial time by a nondeterministic machine M, define $V(\cdot,\cdot)$ that takes as input a value x that would be given as input to M and a description of a computational path of $M(x)$ leading to acceptance. V accepts if this description is valid, and runs in polynomial time. Thus $L \in \mathbf{NP}$, establishing our result. □

3 Space-Bounded Complexity Classes

A machine solves a problem using space $s(\cdot)$ if for every input x, the machine outputs the correct answer and uses only the first $s(|x|)$ cells of the tape. For a standard Turing machine, we can't do better than linear space since x itself must be on the tape! So we will often consider a machine with multiple tapes: a read-only "input" tape, a read/write "work" or "memory" tape, and possibly a write-once "output" tape. Then we can say the machine uses space s if for input x, it uses only the first $s(|x|)$ cells of the work tape.

We denote by ł the set of decision problems solvable in $O(\log n)$ space. We denote by **PSPACE** the set of decision problems solvable in polynomial space.

Theorem 3
If a machine always halts, and uses $s(\cdot)$ space, with $s(n) \geq \log n$, then it runs in time $2^{O(s(n))}$.

Proof: Call the "configuration" of a machine M on input x a description of the state of M, the position of the input tape, and the contents of the work tape at a given time. Write down $c_1, c_2, \ldots, c_t$ where c_i is the configuration at time i and t is the running time of $M(x)$. No two c_i can be equal, or else the machine would be in a loop, since the c_i completely describes the present, and therefore the future, of the computation! Now the number of *possible* configurations is simply the product of the number of states, the number of positions on the input tape, and the number of possible contents of the work tape (which itself depends on the number of allowable positions on the input tape). This is

$$O(1) \cdot n \cdot |\Sigma|^{s(n)} = 2^{O(s(n))+\log n} = 2^{O(s(n))}$$

Since we cannot visit a configuration twice during the computation, the computation must therefore finish in $2^{O(s(n))}$ steps. $\square$

NL is the set of decision problems solvable by a non-deterministic machine using $O(\log n)$ space. **NPSPACE** is the set of decision problems solvable by a non-deterministic machine using polynomial space.

Analogously with time-bounded complexity classes, we could think that **NL** is exactly the set of decision problems that have "solutions" that can verified in log-space. If so, **NL** would be equal to **NP**, since there is a log-space algorithm V that verifies solutions to SAT. However, this is unlikely to be true, because of the surprising fact that **NL** is contained in **P**! Intuitively, not all problems with a log-space "verifier" can be simulated in **NL**, because an **NL** machine does not have enough memory to keep track of all the choices that it makes.

Theorem 4
$\mathbf{NL} \subseteq \mathbf{P}$.

Proof: Let L be a language in **NL** and let M be a non-deterministic log-space machine for L. Consider a computation of $M(x)$. As before, at any time there are $2^{O(s(n))} = n^{O(1)}$ possible configurations. Consider a directed graph in which vertices are configurations and edges indicate transitions from one state to another which the machine is allowed to make in a single step (as determined by its δ). This graph has polynomially many vertices, so in polynomial time we can do a depth-first search to see whether there is a path from the initial configuration that eventually leads to acceptance. This describes a polynomial-time algorithm for deciding L, so we're done. $\square$