CSE 531: Computational Complexity I **Winter 2016**

Lecture 3: Nondeterminism, NP, and NP-completeness

January 13, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 Nondeterminism and NP

Recall the definition from last class:

Definition 1.1. NP *is the set of languages $L \subseteq \{0,1\}^*$ such that there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM V (called a* verifier*) such that for all $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} V(x,y) = 1.$$

Such a y is called a certificate/witness/proof *that $x \in L$.*

We continue with our list of NP problems.

Some more NP problems

- $INDEPENDENT\text{-}SET = \{[G,k] \ : \ G$ is an undirected graph with an independent set of vertices $I \subseteq V(G)$ of size $\geq k\}$ where a subset I is independent iff it contains no edges of G.
  - Guess a set I of size k. Verify that it contains no edges of G.
- $CLIQUE = \{[G,k] \ : \ G$ is an undirected graph with an independent set of vertices $C \subseteq V(G)$ of size $\geq k\}$ where a subset C is a clique iff every edge on the vertices of C are in G.
  - Guess a set C of size k. Verify that all edges in C are present in G.
- $COMPOSITE = \{[n] \ : \ n$ is a composite integer$\}$.
  - Guess a set integers a and b. Verify that $1 < a, b < n$ and $ab = n$.
- $USTCONN = \{[G,s,t] \ : \ G$ is an undirected path from s to $t\}$.
  - Guess the path. Verify that it connects s and t.

- $GRAPH\text{-}ISOMORPHISM = \{[G, H] \ : \ G$ and H are undirected graphs such that $|V(G)| = |V(H)|$ and there is a 1-1, onto map $\phi : V(G) \to V(H)$ satisfying $(\phi(u), \phi(v)) \in E(H)$ iff $(u, v) \in E(G)$ for all $(u, v) \in V(G)$.
  - Guess the function ϕ. Verify that it is an isomorphism.

We note that $USTCONN$ has long been known to also be in P, $COMPOSITE$ was only shown to be in P in 2005 (by Reingold), and just a month ago $GRAPH\text{-}ISOMORPHISM$, which previously had no algorithm for efficient than factoring in the worst case, was shown by Babai to be computable in $\mathsf{DTIME}(\mathsf{n}^{\log^c \mathsf{n}})$ for some constant c.

Analogous to polynomial time $\mathsf{P} = \bigcup_{\mathsf{k}} \mathsf{DTIME}(\mathsf{n}^{\mathsf{k}})$ one can define exponential time

$$\mathsf{EXP} = \bigcup_{\mathsf{k}} \mathsf{DTIME}(2^{\mathsf{n}^{\mathsf{k}}}).$$

Lemma 1.2. $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$.

Proof. We already saw that $\mathsf{P} \subseteq \mathsf{NP}$. To show that $\mathsf{NP} \subseteq \mathsf{EXP}$ we use brute force: Let $L \in \mathsf{NP}$. Then there is some running time p and polynomial time verifier TM V such that $x \in L$ iff there is a $y \in \{0,1\}^{p(|x|)}$ such that $V(x, y) = 1$. The algorithm is as follows: For each $y \in \{0,1\}^{p(|x|)}$, run V on input (x, y) and immediately accept if V accepts any input; otherwise, reject. The running time is $2^{p(|x|)}q(|x|, p(|x|))$ where q is the running time bound for V. Now $p(|x|)$ is $O(|x|^k)$ for some integer k and since q is also a polynomial, the total running time is $O(2^{n^{k+1}})$ and hence $L \in \mathsf{EXP}$ as required. □

Nondeterministic Turing machines The original definition of NP was in terms of *nondeterministic* Turing machines. Nondeterministic Turing machines (NTMs) are formally defined the same as determinstic ones except that the transition function δ gives a *set* of possible transitions for each step, rather than just a single one. That is

$$\delta : Q \times \Gamma^{k+1} \to \mathcal{P}(Q \times \Gamma^{k+1} \times \{L, S, R\}^{k+1}).$$

For the computation of decision problems, we use a special q_{accept} state in lieu of an output tape in lieu of writing a 0 or 1 on the output tape.

Recall that a *configuration* of a TM or NTM consists of its state, tape contents, and the positions of its read/write heads. A (deterministic) TM computation can be thought of as a path in which each node is a configuration and each computation step is a directed edge. (If the computation loops then this path will close on itself but for computation bounded by time $T(|x|)$ it will remain a path of length $T(|x|)$.)

For an NTM M, each configuration may have many possible next configurations, so each node has out-degree at most $B = |Q \times \Gamma^{k+1} \times \{L, S, R\}^{k+1}|$ so we can describe computations as a

rooted tree of height at most $T(|x|)$ (where we ignore the fact that the same configuration may be reachable along different branches). Note that some branches may end early (fail) in case the δ function has output $\emptyset$.

Definition 1.3. *An NTM M accepts an input x (outputs 1) iff there is a sequence of possible moves on input x that lead M to q_{accept}. (That is, there is some leaf of the computation tree of M on input x that is a configuration in state q_{accept}.)*

An NTM M computes a function $f : \{0,1\}^* \to \{0,1\}^*$ iff
(1) the output on all non-failing computation is $f(x)$, and
(2) there is at least one path that outputs $f(x)$.
This can be seen as a generalization of the case of decision problems.

The time taken by an NTM M on input x is the maximum number of steps of any possible computation of M on input x. We then define nondeterministic time complexity classes analogously to the deterministic ones.

Definition 1.4. *We define the complexity class of Boolean functions,*

$$\mathsf{NTIME}(\mathsf{T}(\mathsf{n})) = \{\mathsf{f} : \{0,1\}^* \to \{0,1\} \mid \textit{there is a NTM } M \textit{ that computes } f \textit{ in time } O(T(n))\}.$$

We also interpret $\mathsf{NTIME}(\mathsf{T}(\mathsf{n}))$ *as a set of languages $L \subseteq \{0,1\}^*$ where $L = \{x \mid M \text{ accepts } x\}$.*

Theorem 1.5. $\mathsf{NP} = \bigcup_{\mathsf{k}} \mathsf{NTIME}(\mathsf{n}^{\mathsf{k}})$

Proof. Let $L \in \mathsf{NP}$. Then there is a polynomial p and a a polynomial-time verifier V such that $x \in L$ iff there is some $y \in \{0,1\}^{p(|x|)}$ and $V(x,y) = 1$. On input x, the NTM M will write down a sequence of bits y of length $p(|x|)$, where the nondeterminism allows each bit to be either 0 or 1. Then M runs V on input (x,y). The running time of M on input x is at most $p(|x|)+q(|x|+p(|x|))$ where q is the running time of V. Since p and q are polynomials, there is some k such that this time is $O(|x|^k)$. (We say that M nondeterministically "guesses" y and then deterministically verifies y.) Therefore $L \in \mathsf{NTIME}(\mathsf{n}^{\mathsf{k}})$.

Let $L \in \mathsf{NTIME}(\mathsf{n}^{\mathsf{k}})$. Then there is an NTM M and a time bound T that is $O(n^k)$ such that M accepts x iff $x \in L$. We define the polynomial p_L and verifier V_L as follows. Let $b = \log_2 B$. Define $p(n) = bT(n)$. The certificate $y \in \{0,1\}^{p(|x|)}$ is interpreted as a sequence of moves of M of length $T(|x|)$.

V on input x and y simulated M on input x using each segment of y of length b to choose which of the possible next moves to execute. If the move is not one of the possible moves then computation stops and outputs 0. If the move yields state q_{accept} of M then V outputs 1. The simulation takes time $O(T(n))$ which is $O(n^k)$. Clearly $x \in L$ iff $V(x,y) = 1$. Hence $L \in \mathsf{NP}$. □

The second direction of the above proof shows that there is a normal form for nondeterministic computations in which a stage of guessing is following by deterministic verification.

We can also define nondeterministic complexity classes for other time bounds, e.g., $\mathsf{NEXP} = \bigcup \mathsf{NTIME}(2^{n^k})$.

2 Polynomial-time reductions and NP-completeness

Definition 2.1. $A \subseteq \{0,1\}^*$ *is* polynomial-time mapping reducible *to* $B \subseteq \{0,1\}^*$, $A \leq_p B$, *iff there is a polynomial-time computable* $f : \{0,1\}^* \to \{0,1\}^*$ *such that for all* $x \in \{0,1\}*$,

$$x \in A \Leftrightarrow f(x) \in B.$$

Lemma 2.2.
(1) If $A \leq_p B$ *and* $B \leq_p C$ *then* $A \leq_p C$,
(2) if $B \in \mathsf{P}$ *and* $A \leq_p B$ *then* $A \in \mathsf{P}$, *and*
(3) if $B \in \mathsf{NP}$ *and* $A \leq_p B$ *then* $A \in \mathsf{NP}$.

Proof.
(1) Let f be the reduction from A to B computable in time $O(n^k)$ and g be the reduction from B to C computable in time $O(n^\ell)$. The reduction h from A to C is $h = g \circ f$. By definition $x \in A \Leftrightarrow f(x) \in B$ and $f(x) \in B \Leftrightarrow g(f(x)) \in C$ so $x \in A \Leftrightarrow h(x) \in C$. It remains to determine the computation time for h. The computation time for $f(x)$ is $O(|x|^k)$. This also implies that $|f(x)|$ if $O(|x|^k)$ and so the computation time of g on input $f(x)$ is $O((|x|^k)^\ell) = O(n^{k\ell})$ it follows that the total time to compute $h(x)$ is $O(n^k + n^{k\ell}) = O(n^{k\ell})$ and hence h is a polynomial-time reduction from A to C as required.
(2) The algorithm on input x is to compute $f(x)$ and run the polynomial-time algorithm for B on input $f(x)$ and take its answer. Correctness is immediate since f is reduction. The time analysis is similar to that for (1).
(3) The verifier V_A for A will take x and a string y of length $p_B(|f(x)|)$ and run the verifier V_B for B on input $(f(x), y)$. Again, the time analysis is similar. (Alternatively, one could use polynomial-time NTMs.) □

Definition 2.3.
(1) B *is* NP-hard *iff* $\forall A \in \mathsf{NP}$, $A \leq_p B$.
(2) B *is* NP-complete *iff (a)* $B \in \mathsf{NP}$ *and (b)* B *is* NP*-hard.*

Lemma 2.4.
(a) If B *is* NP*-hard and* $B \in \mathsf{P}$ *then* $\mathsf{P} = \mathsf{NP}$.
(b) $\mathsf{P} = \mathsf{NP}$ *iff some* NP*-complete problem is in* P.

Proof.
(a) Let $A \in \mathsf{NP}$. Since B is NP-hard then $A \leq_p B$ and since $B \in \mathsf{P}$ by Lemma 2.2 we have $A \in \mathsf{P}$. It follows that $\mathsf{NP} \subset \mathsf{P}$. Since we already know that $\mathsf{P} \subseteq \mathsf{NP}$, we have $\mathsf{P} = \mathsf{NP}$.

(b) Any NP-complete problem is in NP so if P = NP then every NP-complete problem is in P. For the reverse direction, observe that every NP-complete problem is NP-hard and so the reverse direction follows from part (a). □

In his original paper on the completeness of the SAT problem for NP, Cook used a more general notion of reduction, related to one used by Turing, in which one can make a polynomial number of subroutine calls to solutions to B in order to determine the answer to A, and one can even modify the answers; these reductions are known as "Cook reductions". In his follow-on paper, Karp refine the notion of reduction to the one we use here; these are also known as "Karp reductions". Though the results of Lemma 2.4 hold for Cook reductions, one difference between the definitions is that Lemma 2.2 part (3) does not hold for Cook reductions; i.e. NP is not closed under Cook reductions.

Theorem 2.5 (Cook-Levin Theorem)**.** $CIRCUIT\text{-}SAT$ *is* NP*-complete.*

Proof. We have already shown that $CIRCUIT\text{-}SAT$ is in NP. It remains to show that it is NP-hard.
Let $A \in$ NP. By definition there are p and V_A such that

$$x \in A \Leftrightarrow V_A(x, y) = 1 \text{ for some } y \in \{0,1\}^{p(|x|)}.$$

Since p is a polynomial and V_A runs in polynomial time the the total running time of V_A on input (x, y) is $O(|x|^k)$ for some integer k. Let M_x be the TM V_A with x hardwired, i.e. $M_x(y) = V_A(x, y)$.

Then by the circuit simulation of Turing machine computations from last class there is a circuit C_x^A of size $O(|x|^k \log |x|)$ that takes y as input, such that $C_x^A(y) = M_x(y) = V_A(x, y) = 1$. Therefore

$$\begin{aligned} C_x^A \in CIRCUIT\text{-}SAT &\Leftrightarrow \exists y C_x^A(y) = 1 \qquad \text{by definition} \\ &\Leftrightarrow \exists y\, V(x, y) = 1 \\ &\Leftrightarrow x \in A \end{aligned}$$

This gives correctness. As we discussed in the proof of the circuit simulation, the circuit is not only polynomial size, it is very regular and easy to compute given x with the transition table of V_A built in. Therefore the map from x to $[C_x^A]$ is polynomial-time computable and hence $A \leq_p CIRCUIT\text{-}SAT$. □

Observe that by transitivity of $\leq_p$ we immediately have

Lemma 2.6. *If* B *is* NP*-hard and* $A \leq B$ *then* C *is* NP*-hard.*

Corollary 2.7 (Cook-Levin)**.** $3SAT$ *is* NP*-complete.*

Proof. We already know that $3SAT \in \mathsf{NP}$. To complete the proof we show that $CIRCUIT\text{-}SAT \leq_p 3SAT$:
The mapping from $[C]$ for some circuit C to $[\phi]$ for a 3CNF formula ϕ is given by the following. We add a new variable for each gate g of C. We first add extra clauses depending on the gate type.

If the gate is $f = g \vee h$ we add clauses for the formula $f \leftrightarrow (g \vee h)$, which are $\neg f \vee g \vee h$, $\neg g \vee f$, and $\neg h \vee f$.

If the gate is $f = g \wedge h$ we add clauses for the formula $f \leftrightarrow (g \wedge h)$, which are $\neg f \vee g$, $\neg f \vee h$, and $\neg g \vee \neg h \vee f$.

If the gate is $f = \neg g$ we add clauses for the formula $f \leftrightarrow \neg g$, which are $\neg f \vee \neg g$, $f \vee g$.

The we add a clause of size 1 consisting of the clause for the output gate.

Finally for every clause of size < 3 we add one or two extra dummy variables w and z and replace by size 3 clauses as follows: $(a \vee b)$ becomes the two clauses $(a \vee b \vee z)(a \vee b \vee \neg z)$ and clause a becomes $(a \vee w \vee z)(a \vee w \vee \neg z)(a \vee \neg w \vee z)(a \vee \neg w \vee \neg z)$.

The reduction is clearly polynomial time, the formula ϕ is satisfied if and only if the output is correctly computed and it has value 1. □

Cook's original paper showed NP-completeness directly for $CNFSAT$ and then $3SAT$. That of Levin used a tiling problem as the first key problem. In both cases, the reduction used was the simpler quadratic size reduction with a tableau having one entry for each tape cell for each time step.

Even in the beginning satisfiability problems were not the only kinds of problems shown to be NP-complete. For example we have the following:

Theorem 2.8. $INDEPENDENT\text{-}SET$ *is* NP*-complete.*

Proof. Again we know that it is in NP. To show NP-hardness we prove

$$3SAT \leq_p INDEPENDENT\text{-}SET :$$

Suppose that the 3CNF formula ϕ has n variables and m clauses $C_1, \ldots, C_m$. The reduction maps ϕ to a $[G, k]$ where G is an undirected graph on $7m$ vertices and $k = m$. There will be 7 vertices associated with each clause C_i of m, one for each of the 7 partial assignments to the 3 variables in C_i that make C_i true. There will be an edge between vertices if and only if their partial assignments disagree. (Note that this means that every pair of vertices associated with a single C_i is joined by an edge.) Clearly this is polynomial time. The graph has size $O(m^2)$ edges.

Given a satisfying assignment α for ϕ, for each C_i we select the unique assignemtn associated with C_i that is consistent with α. Hence $[\phi] \in 3SAT$ implies $[G, m] \in INDEPENDENT\text{-}SET$.

On the other hand an independent set in G of size m must contain precisely 1 of the 7 vertices associated with each C_i. Since they form an independent set, by definition, they agree on their common variables. The satisfying assignment for ϕ is the (unique) assignment that is consistent with all of the vertices in the independent set. $\square$