CSE200 Lecture Notes
The class NP

Lecture by Russell Impagliazzo
Notes by William Matthews

Lecture April 21, 2010

Let E stand for "exponential time", $DTIME(2^{n^{O(1)}})$, and EE stand for "doubly exponential time", $DTIME(2^{2^{n^{O(1)}}})$. We know thus far that $P \subset E \subset EE \subset \cdots \subset REC \subset RE$. Thus, we know that there are many many very hard problems, but are there hard problems that we really care about?

"90%" of CS problems are search or optimization.

Search: Find a solution that meets a set of constraints determined by the instance.

Optimization: Find a solution that meets the constraints and is the best according to an objective functions.

In such problems, the search space is large but bounded in terms of the instances.

Examples: Search: Given a protocol, does it satisfy some security property on every run? – Is there a run that violates the security constraint? Optimization: Given a circuit to layout on a chip, what's the best way to organize the components (subject to various physical constraints)?

We would like to say that it's possible but difficult, in some sense, to solve a search problem. We will characterize search problems as as an instance x, a solution format y, and a constraint relation $c(x, y) \in \{0, 1\}$. Since we think of polynomial time as "efficient", we will think of a search problem as reasonable if (i) the solution can be represented as a polynomial length string ($|y| \leq poly(|x|)$), and (ii) given x and y, we can compute $c(x, y)$ in time $poly(|x|)$ (which is also $poly(|x|, |y|)$.) We formalize this notion by defining the class $Search_{c,f}$ as given x, find a y such that $|y| \leq f(|x|)$ and $c(x, y)$ if such a y exists, and define the class $Search - P$ as the class of all $Search_{c,f}$ for $c \in P$ and $f \in n^{O(1)}$.

Example: 3-coloring (search problem): An instance is a graph $G = (V, E)$ an a solution is a map $\chi : V \to \{\mathbf{R}, \mathbf{G}, \mathbf{B}\}$, and the constraint is that for each edge $\{u, v\} \in E$, $\chi(u) \neq \chi(v)$. Given G, can we find χ?

For each search problem, we can also consider the corresponding decision problem: For each instance rather than finding a solution, we only ask whether a solution exists.

Example: 3-coloring decision problem: An instance is a graph $G = (V, E)$ an a solution is a map $\chi : V \to \{\mathbf{R}, \mathbf{G}, \mathbf{B}\}$, and the constraint is that for each edge $\{u, v\} \in E$, $\chi(u) \neq \chi(v)$. Given G, *does there exist* a χ?

Formally, define the class NP of decision problems corresponding to search problems in $Search - P$. One of the big open problems is does $P = NP$? Every language in P is also in NP since in the constraint c may just decide in polynomial time whether the instance is in the language.

What if $P = NP$, and what if $P \neq NP$? If $P = NP$, then the world is strange and scary!!! (To be made formal later on!) If $P \neq NP$ then specific problems (that will often appear in real work) are not in P.

$NP \in DTIME(2^{poly(n)})$ since that's the amount of time needed to enumerate all possible solutions and check each in polynomial time. Thus $P \subseteq NP \subseteq E$.

Why "NP?" The N stands for *non-deterministic*. In an (standard / deterministic) TM, the transition function deterministically specifies a single action determined by the state of the TM and the symbols under the tape head(s). In a *non-deterministic* TM, the transition function specifies a set of possible actions determined by the state of the TM and the symbols under the tape head(s). Given a non-deterministic TM

M and input x, what do the set of computations of M on x look like? Since at each step, we may have many possible actions so there may be exponentially many possible different computations. We will say that a non-deterministic TM (NTM) M accepts x if there exists a run of M on x which accepts (even if there are many possible computations which reject). A NTM M runs in non-deterministic time T if every run of x terminates within $T(|x|)$ steps. Define the class $NTIME(T(n))$ of languages L decided by a NTM which runs in time at most $T(n)$. Claim: $NP = \cup_k NTIME(n^k)$.

Proof: (Old definition implies new definition.) For $L \in NP$, there exists $c \in P$ such that $x \in L \iff \exists y, |y| \leq poly(|x|)$ and $c(x, y)$. We claim the following NTM M decides L: Write "$(x$," to the tape. For $poly(|x|)$ steps, nondeterministically either write "0" or "1" and move right. Write ")". Note that we now have (x, y) for some non-deterministically "guessed" y on the tape. Now run c and accept iff c accepts. There exists a y which causes c to accept if and only if at some point we non-deterministically write it and the run c, which causes M to accept.

(New definition implies old definition.) Assume L is decided by a NTM M in $poly(n)$ steps. View a solution y as a sequence of non-deterministic actions for M. Since there are a bounded possible number of actions for M, and $poly(n)$ time steps, we get need at most a polynomial length y. For the constraint c, we simulate M on input x, and whenever we need to make a non-deterministic choice of action we choose an action based on the next action in y. c will accept iff only if the sequence of non-deterministic choices encoded in y causes M to accept. If an instance $x \in L$ then there exists an accepting computation path for M, from which we can know there exists a solution y. Any solution y which causes c to accept must correspond to an accepting computation path for M.

Assuming that $P \neq NP$, what can we say about "hard" problems in NP?

Recall that a mapping reduction from a language L_1 to a language L_2 is a computable function f such that $x \in L_1 \iff f(x) \in L_2$. If such a f exists then $L_1 \leq_m L_2$. If, in addition, $f \in FP$ (f is computable in polynomial time) then we say that f is a polynomial time mapping reduction and write $L_1 \leq_{pm} L_2$. (If $L_1 \leq_{pm} L_2$ and $L_2 \in P$ then $L_1 \in P$, and $\leq_{pm}$ is a transitive relation: $L_1 \leq_{pm} L_2$ and $L_2 \leq_{pm} L_3$ then $L_1 \leq_{pm} L_3$.)

L is hard for NP (written NP-hard) if for all $L' \in NP$, $L' \leq_{pm} L$. L is complete for NP (written NP-complete) if L is NP-hard and $L \in NP$. For example, the halting problem is NP-hard, but this isn't a terribly interesting fact. More interesting is the question of whether there exist NP-complete problems.

Consider the NP problem $CIRCUIT - SAT$: An instance is a circuit $C(y_1, \ldots, y_n)$, a solution is an assignment $y_1 = a_1, \ldots, y_n = a_n$, and the constraint is that $C(a_1, \ldots, a_n)$.

We claim that $CIRCUIT - SAT$ is NP-complete. For any L in NP, there exists a polynomial time computable relation R such that $x \in L \iff \exists y$ such that $R(x, y)$. We give the following reduction from L to $CIRCUIT - SAT$. The reduction $f(x)$ does the following on input x: Construct a circuit $C_0(x_1, \ldots, x_n, y_1, \ldots, y_m)$ that simulates $R(x, y)$. Set $x_1, \ldots, x_n$ to the bits of the input x to give the circuit $C_1(y_1, \ldots, y_m)$. Output C_1. The validity of the reduction more or less follows from the construction.

Thus, for all $L \in NP$ we have $L \leq_{pm} CIRCUIT-SAT$ so $CIRCUIT-SAT \in NP$-hard. Verifying that $CIRCUIT - SAT \in NP$ is straightforward. So we can conclude that $CIRCUIT - SAT \in NP$-complete.