

CMPT 710 - Complexity Theory: Lecture 10

Valentine Kabanets

October 4, 2007

1 “Search-to-Decision” Reductions

Suppose that $P = NP$. That would mean that all NP languages can be decided in deterministic polytime. For example, given a graph, we could decide in deterministic polytime whether that graph is 3-colorable. But could we find an actual 3-coloring? It turns out that yes, we can.

In general, we can define an **NP search problem**: Given a polytime relation R , a constant c , and a string x , find a string y , $|y| \leq |x|^c$, such that $R(x, y)$ is true, if such a y exists. . As the following theorem shows, if $P = NP$, then every NP search problem can also be solved in deterministic polytime.

Theorem 1. *If $NP = P$, then there is a deterministic polytime algorithm that, given a formula $\phi(y_1, \dots, y_n)$, finds a satisfying assignment to ϕ , if such an assignment exists.*

Proof. We use a kind of binary search to look for a satisfying assignment to ϕ . First, we check if $\phi(x_1, \dots, x_n) \in SAT$. Since we assumed that $P = NP$, this can be done in deterministic polytime. Then we check if $\phi(0, x_2, \dots, x_n) \in SAT$, i.e., if ϕ with x_1 set to False is still satisfiable. If it is, then we set a_1 to be 0; otherwise, we make $a_1 = 1$. In the next step, we check if $\phi(a_1, 0, x_3, \dots, x_n) \in SAT$. If it is, we set $a_2 = 0$; otherwise, we set $a_2 = 1$. We continue this way for n steps. By the end, we have a complete assignment $a_1, \dots, a_n$ to variables $x_1, \dots, x_n$, and by construction, this assignment must be satisfying.

The amount of time our algorithm takes is polynomial in the size of ϕ : we have n steps, where at each step we must answer a SAT question. Since, by our assumption, $P = NP$, each step takes polytime. $\square$

Theorem 1 shows the true importance of proving that $NP = P$. If $NP = P$, we could efficiently generate a correct solution for any problem with an efficient recognition algorithm for correct solution. For instance, if $P = NP$, then we could efficiently find a login password of any user of a network, since checking if a password matches a login name can be done efficiently. Thus, if $P = NP$, essentially any secret could be found out efficiently.

As another example of the “search-to-decision” reduction, consider the problem Hamiltonian Cycle: Given an undirected graph G , decide if G has a Hamiltonian cycle (i.e., a cycle that visits every vertex of G exactly once). The corresponding search problem is: Given a graph G , find a Hamiltonian cycle in G , if such a cycle exists.

Assuming that we have access to a subroutine solving the decision version of Hamiltonian Cycle, here is an efficient algorithm for solving the search version: If G has no Hamiltonian cycle, then output “No” and halt. Otherwise, for each edge e of the graph G , if $G - e$ has a Hamiltonian cycle then $G = G - e$. After all the edges have been checked, the remaining graph is exactly a Hamiltonian cycle of G .

It should be stressed that we are interested in *efficient* (i.e., polytime) search-to-decision reductions. Such efficient reductions allow us to say that if the decision version of our problem is in P, then there is also a polytime algorithm solving the corresponding search version of the problem.

2 Nondeterministic Time Hierarchy

We want to argue that in more nondeterministic time, we can accept more languages. Recall how we argued that in the case of *deterministic* Turing machines. Given a proper complexity function $t(n)$, we constructed a language $Diag_{t(n)}$ that cannot be in $\text{Time}(t(n))$ by “diagonalizing” against every deterministic TM running in time $t(n)$. That is, we considered an enumeration of all TM’s $M_1, M_2, \dots, M_i, \dots$ and all inputs $x_1, x_2, \dots, x_i, \dots$, and defined

$$Diag_{t(n)} = \{x_i \mid M_i \text{ does not accept } x_i \text{ in } t(|x_i|) \text{ steps}\}$$

Then we argued that

1. The language $Diag_{t(n)}$ is not in $\text{Time}(t(n))$ (since it differs from the language of any $t(n)$ -time TM on at least one input).
2. The language $Diag_{t(n)}$ is in $\text{Time}(t^3(n))$ (since we can simulate a deterministic TM on a given input, and then flip its answer).

For the case of nondeterministic TM’s, we may try to follow the same approach. We can define

$$NDiag_{t(n)} = \{x_i \mid \text{NTM } M_i \text{ does not accept } x_i \text{ in } t(|x_i|) \text{ steps}\}$$

As before, it is possible to show (with exactly the same proof as in the Time case) that the new language $NDiag_{t(n)}$ is not in $\text{NTIME}(t(n))$. But, it is not at all clear if $NDiag_{t(n)}$ is in $\text{NTIME}(t^c(n))$ for some constant c . The difficulty is that, unlike the case of *deterministic* TM’s, we cannot flip the answers of a NTM deciding language L to get an NTM deciding the complement of L . This is related to the big open question $\text{NP} \stackrel{?}{=} \text{coNP}$.

Therefore, we must use a different approach. It is still based on diagonalization, but a different kind of diagonalization - so-called “lazy” diagonalization.

Theorem 2 (NTIME Hierarchy Theorem). *For every proper complexity function $f(n) \geq n$ and $g(n) \in \omega(f(n+1))$, we have*

$$\text{NTIME}(f(n)) \subsetneq \text{NTIME}(g(n)).$$

Proof. First of all, we will prove the theorem for the case of *unary* input alphabets, i.e., our NTMs will have 1^n as inputs. (This makes the theorem stronger.)

Let $t(n)$ be a sufficiently fast growing function so that a deterministic $t(n)$ -time TM can decide if a nondeterministic $f(n)$ -time TM accepts unary input 1^n . (Think of $t(n) > 2^{f(n)}$.)

We will define a NTM D whose language differs from every $L(M_i)$, where M_i is an i th NTM clocked to run for at most $f(n)$ steps. Our NTM D will diagonalize against each M_i as follows. Let us partition the interval $[1..\infty)$ of natural numbers into a collection of finite subintervals: $[1..t(1)]$, $[t(1) + 1..t(t(1))]$, $\dots$, $[t^{(i-1)}(1) + 1..t^{(i)}(1)]$, $\dots$, where $t^{(i)}$ denotes the composition of t with itself i times. Let us number these intervals $1, 2, \dots$ so that $[t^{(i-1)}(1) + 1..t^{(i)}(1)]$ is the i th interval.

We will use the i th interval to diagonalize against NTM M_i . For notational convenience, let the i th interval be $[k..n]$. We define the behaviour of NTM D as follows:

1. for $k \leq j < n$, $D(1^j)$ accepts iff $M_i(1^{j+1})$ accepts;
2. $D(1^n)$ accepts iff $M_i(1^k)$ does not accept.

Let us see how such a definition of D diagonalizes against M_i . Suppose that $L(D) = L(M_i)$. This and part (1) of the definition of D ensures that D accepts 1^k iff D accepts 1^n . But then part (2) of the definition of D ensures that D accepts 1^n iff D does not accept 1^k . A contradiction. Hence, $L(D) \neq L(M_i)$.

Now let us analyze how much time D needs on input 1^j . We need to compute which interval $i = [k..n]$ contains j (so that we know which machine to diagonalize against); this computation is efficient for “good” $t(n)$ that we chose. Now, depending on where in the interval j is, we either need (if $k \leq j < n$) to simulate $M_i(1^{j+1})$ nondeterministically, in time $O(f(j+1))$, or (if $j = n$) deterministically simulate $M_i(1^k)$ in time j . So, in total, $D(1^j)$ runs in time at most $g(j)$. $\square$

As a corollary of the Nondeterministic Time Hierarchy Theorem, we obtain the following.

Theorem 3. $\text{NP} \subsetneq \text{NEXP}$

Proof. $\text{NP} \subseteq \text{NTime}(2^n) \subsetneq \text{NTime}(2^{n^2}) \subseteq \text{NEXP}$. $\square$