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

Lecture 6: Oracle TMs, Diagonalization Limits, Space Complexity

January 22, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

Diagonalization enabled us to separate many complexity classes. Could it be used to separate P from NP? In order to answer this we visit another stronger notion of reduction due to Turing and Cook.

1 Oracle TMs and P versus NP

Definition 1.1. *An* oracle TM $M^?$ *is a Turing machine that an extra tape, called the oracle tape, as well as three special states, a query state q_{query} and two answer states q_{yes} and q_{no}. It is designed to be used with an* oracle *(or black box) that decides decides membership in some set of strings. When $M^?$ enters the query state with a string z on its query tape, in the next step, the machine enters state q_{yes} or q_{no} depending on whether z is a member of the oracle set. Each oracle query takes one time step. We can have oracle NTMs as well as oracle TMs.*

When the oracle set is a fixed set A, we write M^A to denote the computing device that results.

Oracle TMs give the most general natural notion of reduction between decision problems.

Definition 1.2. *A is* Turing reducible *to B, $A^T \leq B$, iff there is an oracle TM $M^?$ such that M^B decides language A.*

Intuitively, this says that one can decide A using multiple adaptive calls to a subroutine that decides B. Though it is not a focus of the current discussion, we also mention the stronger notion of polynomial-time reduction that results from this point of view.

Definition 1.3. *A is* polynomial-time Turing reducible *to B, $A_P^T \leq B$ iff there is an oracle TM $M^?$ such that M^B decides language A and the running time of M^B is polynomial in its input size. In this case, we also say that A is* Cook-reducible *to B, in contrast to the polynomial-time mapping reductions being known as Karp reductions,*

Cook reductions preserve P, as do Karp reductions, but unlike Karp reductions they do not necessarily preserve NP. The terminology reflects differences between the notions of reduction in Cook's 1971 paper and Karp's follow-on 1972 paper on NP-completeness.

Definition 1.4. *For any $A \subseteq \{0,1\}^*$, define*

- $\mathsf{DTIME^A((T(n))} = \{\mathsf{L} \mid \mathsf{L}$ *is decided by some oracle TM M^A with running time $O(T(n))\}$.*
- $\mathsf{NTIME^A((T(n))} = \{\mathsf{L} \mid \mathsf{L}$ *is decided by some oracle NTM M^A with running time $O(T(n))\}$.*

A key property of diagonalization is that it only involves simulation of Turing machines by other Turing machines. Because of this, it still works equally well with oracle TMs. For example, we have the following:

Theorem 1.5. *Let $f, g : \mathbb{N} \to \mathbb{N}$.*

- *If g is time-constructible and $f(n)log_2 f(n)$ is $o(g(n))$ then for all $A \subseteq \{0,1\}^*$, $\mathsf{DTIME^A(f(n))} \subsetneq \mathsf{DTIME^A(g(n))}$.*
- *If g is time-constructible and $f(n+1)$ is $o(g(n))$ then for all $A \subseteq \{0,1\}^*$, $\mathsf{NTIME^A(f(n))} \subsetneq \mathsf{NTIME^A(g(n))}$.*

Proof. The proofs are identical to those of the ones without the oracle A except that enumerations are of oracle machines $M^?$ rather than ordinary ones and every machine is given oracle A. □

When we have a statement holds that for Turing machines that have an added oracle A if and only if it holds for Turing machines without an oracle, we say that it is true *relative* to A. When it is true relative to oracle A for every oracle A, as in the above theorem, then we say that the statement *relativizes*. Diagonalization arguments always relativize.

However, we now see that the P verus NP question cannot relativize and hence diagonalization arguments will not suffice either to prove that $\mathsf{P} \neq \mathsf{NP}$ and simulation arguments will not be enough to prove $\mathsf{P} = \mathsf{NP}$.

Theorem 1.6 (Baker,Gill,Sollovay)**.** *There are oracles A and B such that* $\mathsf{P^A} = \mathsf{NP^A}$ *but* $\mathsf{P^B} \neq \mathsf{NP^B}$.

In order to prove the first part of the theorem we will need a language that is sufficiently hard that a determinstic solution to it can capure all of NP. For convenience, we will use a generic problem that is complete for EXP. (We haven't defined this yet but it is the obvious analog of NP-complete.

Definition 1.7. *Define $EXPCOM = \{[M, x, 1^n] \mid M$ outputs 1 on input x in at most 2^n steps.$\}$.*

Lemma 1.8. *$EXPCOM$ is* EXP *complete. more precisely,*

1. *$EXPCOM$* $\in \mathsf{DTIME(n2^n)} \subset \mathsf{EXP}$

2. *For all $A \in \mathsf{EXP}$, $A \leq_P EXPCOM$.*

Proof. For the first part observe that we run the universal TM for M on input x and simply add a clock based on the 1^n in the input that counts down from 2^n for every simulated step.

For the second part, consider an arbitrary $A \in \mathsf{EXP}$ and let M_A be an associated TM that decides A in time at most 2^{kn^k} for some integer k. The reduction f from A to $EXPCOM$ takes an input x for A and creates the string $y = [M_A, x, 1^{k|x|^k}]$. This can be done in time $O(n^k)$. Clearly $x \in A$ iff $M_A(x) = 1$ which holds iff $y \in EXPCOM$. $\square$

Proof of Theorem 1.6. We will use $EXPCOM$ for the oracle A. By definition of oracle TMs, $\mathsf{P}^{\mathsf{EXPCOM}} \subseteq \mathsf{NP}^{\mathsf{EXPCOM}}$. Since $EXPCOM$ is complete for EXP we immediately have

$$\mathsf{EXP} \subseteq \mathsf{P^EXPCOM} \subseteq \mathsf{NP}^{\mathsf{EXPCOM}}.$$

We will show that $\mathsf{NP}^{\mathsf{EXPCOM}} \subseteq \mathsf{EXP}$ which will prove that $\mathsf{P}^{\mathsf{EXPCOM}} = \mathsf{NP}^{\mathsf{EXPCOM}}$. Now any language L in $\mathsf{NP}^{\mathsf{EXPCOM}}$ is decided by some oracle NTM $M^?$ running in time $O(n^\ell)$ for some integer ℓ that can make at most one call to $EXPCOM$ per step. We will simulate this deterministically. To do this we enumerate over each of the $2^{O(n^\ell)}$ paths of the NTM computation. Whenever a query is made to the $EXPCOM$ oracle we use the $O(2^n n)$ algorithm for $EXPCOM$ shown in the lemma. The size of the query can be at most $O(n^\ell)$ so each oracle call requires at most time $2^{O(n^\ell)}O(n^\ell)$ and the total per path is only an $O(n^\ell)$ factor larger. Multiplying by the number of paths, this is still $2^{O(n^\ell)}$ time and hence $L \in \mathsf{EXP}$ as required.

In order to construct an oracle B such that $\mathsf{P^B} \neq \mathsf{NP^B}$, we first define a language based on arbitrary oracle B that will be our separating language for the right choice of B. Define

$$L(B) = \{1^n \mid \exists y \in B.\ |y| = n\}.$$

It is easy to see that $L(B) \in \mathsf{N^B}$ for every oracle B: One input x, first check that $x = 1^n$ for some n. Then guess a $y \in \{0,1\}^n$ on the oracle tape and query B. Output 1 if and only if B answers yes.

We now construct a set B such that $L(B) \notin \mathsf{P^B}$. We will in fact construct B such that no deterministic oracle TM $M^?$ running in at most $2^n/2$ steps with oracle B can decide $L(B)$. To do this we will show how to fool each oracle TM on some input in turn. This will feel a little bit like the diagonalization proofs but we will not need to worry about the resources required to build B; we just need to show that B exists.

Let $M_1^?, M_2^?, \ldots$ be an enumeration of all oracle TMs in which each oracle TM appears infinitely often in the list. (We can do this using the $[M]01^k$ coding trick we used before for the hierarchy theorem.) Define $n_1 = 2$ and $n_{i+1} = 2^{n_i}/2 + 1$. We will define B iteratively, starting with $B_0 = \emptyset$ and adding strings as we go to create $B_1, B_2, \ldots$. B will be the limit (union) of this sequence. In general, B will agree with B_i on all strings of length $< n_{i+1}$ B_{i-1} and B_i will only differ on strings of length $\geq n_i$ but $< n_{i+1}$.

Suppose that $B_0, \ldots, B_{i-1}$ have already been defined. Consider the computation of $M_i^{B_{i-1}}$ on input 1^{n_i} for up to $2^{n_i}/2$ steps. On input 1^{n_i}, M_i queries B_{i-1} for at most $2^{n_i}/2$ strings of length n_i so there must be some string $y \in \{0,1\}^n$ that M_i never asks about. We now define B_i as follows:

$$B_i = \begin{cases} B_{i-1} \cup \{y\} & M_i^{B_{i-1}}(1^{n_i}) = 0 \\ B_{i-1} & M_i^{B_{i-1}}(1^{n_i}) = 1. \end{cases}$$

Observe that we have constructed B_i so that $1^{n_i} \in L(B_i)$ if and only if $M_i^{B_{i-1}}$ rejects 1^{n_i}. Also observe that since B_{i-1} and B_i differ only on the string y which M does not query given oracle B_{i-1}, $M_i^{B_i}$ also gives the wrong answer for $L(B_i)$ on input 1^{n_i}. Finally, we have constructed things so that B_i and B are the same for all strings up to length $2^{n_i}/2$ which is the largest size input that $M_i^?$ can query on input 1^{n_i} in time $2^{n_i}/2$, no matter what the oracle. This means that the behavior of M_i^B on input 1^{n_i} is exactly the same as that of $M_i^{B_i}$. Moreover, since B and B_i agree on inputs of length n_i, $1^{n_i} \in L(B)$ if and only if M_i^B does not accept 1^{n_i} in $2^{n_i}/2$ steps.

Therefore no oracle TM M^B running in time $2^n/2$ can decide $L(B)$ and hence $L(B) \notin \mathsf{P^B}$, proving that $\mathsf{P^B} \neq \mathsf{NP^B}$. □

2 The structure of NP

If $\mathsf{P} \neq \mathsf{NP}$ what does the structure of NP look like under polynomial-time mapping reductions?

Definition 2.1. *Write the equivalence relation $A \equiv_P B$ iff $A \leq_P B$ and $B \leq_P A$. Write $A <_P B$ iff $A \leq_P B$ but $A \not\equiv_P B$.*

The following are easy consequences of the definitions.

Proposition 2.2. *(1) If A and B are NP-complete then $A \equiv_P B$. (2) If $A, B \in \mathsf{P}$, $A, B \subseteq \{0,1\}^*$, and and neither A nor B is $\emptyset$ or $\{0,1\}^*$ then $A \equiv_P B$.*

If $\mathsf{P} = \mathsf{NP}$ then the above two equivalence classes are equal. Ladner looked at the question of what happens if $\mathsf{P} \neq \mathsf{NP}$.

He showed the following:

Theorem 2.3 (Ladner). *If* P|neNP *then one can embed every countable partial order $<$ in the order $<_P$ on the set* NP. *In particular, there exist languages L that are neither in* P *nor* NP*-complete.*

A proof of the latter consequence is in the text. The example, as with all of the examples in Ladner's theorem is fairly technical and we will not go over it in class.

3 Space Complexity

We recall the definition of deterministic space complexity classes and add the definition for non-deterministic space complexity classes.

Definition 3.1.

$$\mathsf{DSPACE(S(n))} = \{\mathsf{L} \subseteq \{0,1\}^* \mid \textit{there is a TM } M \textit{ that decides } L \textit{ using at most } O(S(|x|)) \textit{ cells on any work tape on any input } x\}.$$

$$\mathsf{NSPACE(S(n))} = \{\mathsf{L} \subseteq \{0,1\}^* \mid \textit{there is an NTM } M \textit{ that decides } L \textit{ using at most } O(S(|x|)) \textit{ cells on any work tape on any computation path on any input } x\}.$$

We observe that when we are only measuring space complexity, we can always assume without loss of generality that when a space-bounded Turing machine finishes its computation, it has erased its work tapes and its tape heads are always at the left ends of the tapes. We call this a *normal-form* TM.

As with time complexity, ther are some weird space bounds that behave oddly and we need to rule out.

Definition 3.2. *A function $S : \mathbb{N} \to \mathbb{N}$ is* space-constructible *iff $S(n) \geq \log_2 n$ and there is a deterministic TM M using space $O(S(n))$ that takes input x and produces $[S(|x|)]$.*

Observe that, for example, since two-way finite automata have the same power as 1-way finite automata $DSPACE(1) = NSPACE(1) = REGULAR$, since we can encode the constant amount of storage of the Turing machines in the state.

Configuration Graphs As discussed earlier, a *configuration* of a Turing machine M on input x consists of (a) the state of M, (b) the contents of its input and work tapes, and (c) the positions of all its tape heads.

Definition 3.3. *The* configuration graph $G_{M,x}$ *of M on input x, is a directed graph whose vertices are the configurations of M on input x such that there is an edge (C, D) between configurations C and D iff D can follow C after one step of computation of M. The configuration graph has a distinguished start node C_0 which is the starting configuration of M on input x.*

As discussed above, if M is in normal form, which we will typically assume, $G_{M,x}$ will have a distinguished target configuration C_{accept} that will be reached iff M reaches state q_{accept} on input x.

Observe that if M is deterministic then for all inputs x, the configuration graph $G_{M,x}$ has out-degree at most 1. On the other hand, if M is nondeterministic then $G_{M,x}$ may have larger out-degree, though it will always be bounded by some constant B depending on M.

Lemma 3.4. *If M uses space $O(S(n))$ then $G_{M,x}$ has $n \cdot 2^{O(S(n))}$ vertices. Moreover, if $S(n) \geq \log_2 n$ then there is a binary representation $[C]$ of the vertices of $G_{M,x}$ using $O(S(n))$ bits per vertex and a CNF formula $\varphi_{M,x}$ of size $O(S(n))$ such that $\varphi_{M,x}([C],[D]) = 1$ if and only if $(C,D) \in E(G_{M,x})$.*

Proof. We begin by counting the configurations of M on input x: There are $O(1)$ possibilities for the state of the M. The contents of the input tape for all these configurations are fixed to x. There are n possibilities for the head position on the input tape. For each of the k work tapes there are $|\Gamma|^{O(S(n))}$ choices of tape contents and $O(S(n))$ choices of head position for each tape. Since the number of tapes k is constant, we get

$$O(1) \cdot n \cdot (|\Gamma|^{O(S(n))} \cdot O(S(n)))^k = n \cdot 2^{O(S(n))}$$

different configurations.

For $S(n) \geq \log_2 n$ this is $2^{O(S(n))}$. Each configuration can be easily laid out as a binary string. Moreover, the existence of the CNF formula comes from checking each pair of rows in the tableau proofs that showed the NP-completeness of SAT. $\square$

Theorem 3.5. *For all space-constructible $S(n) \geq \log_2 n$,*

$$\mathsf{NTIME(S(n))} \subseteq \mathsf{DSPACE(S(n))} \subseteq \mathsf{NSPACE(S(n))} \subseteq \mathsf{DTIME(2^{O(S(n))})},$$

where $\mathsf{DTIME(2^{O(S(n))})} = \bigcup_{\mathsf{c}} \mathsf{DTIME(2^{cS(n)})}$.

Proof. The middle containment is immediate by definition.

We begin with the right-hand containment. Let $L \in \mathsf{NSPACE(S(n))}$ and let M be a space $O(S(n))$ NTM that decides L. Note that by definition $G_{M,x}$ is a directed graph with $2^{O(S(|x|))}$ nodes and M accepts x if and only if there is a path in $G_{M,x}$ from C_0 to C_{accept}. The deterministic algorithm on input x will run a standard graph search algorithm such as BFS or DFS starting at C_0 to see if C_{accept} is reachable. This runs in time linear in the number of edges and vertices of $G_{M,x}$ which is $2^{O(S(n))}$ time.

For the left-hand containment, suppose that $L' \in \mathsf{NTIME(S(n))}$ and let M' be an NTM with time bound $O(S(n))$ that decides L'. The deterministic simulation of M' on input x will iterate over each of the $2^{O(S(|x|))}$ paths in the computation of NTM M'. (These can be described by the sequences in the set $[B]^{O(S(|x|))}$ that select the chosen path.) For each such sequence, the deterministic TM will execute that sequence in time $O(S(|x|))$, which can touch at most $O(S(n))$ memory cells. The work tapes store the current element of $[B]^{O(S(n))}$ and the current contents of the work tapes of NTM M'. Therefore it runs in space $O(S(n))$ as required. $\square$