

Lecture Notes 5:**Decision vs. Search, Hierarchy Theorems****Reading.**

- Arora-Barak § 2.5, 3.1-3.2, 4.1.3

Last time: NP-Completeness, Cook-Levin Theorem

If you have extra time after the quiz, try to prove the following:

Theorem 1. *There exists a universal nondeterministic TM U such that, for every α (representing an NTM) and x , we have $U(x, \alpha) = N_\alpha(x)$ and, if N_α halts on x within T steps (on every branch of computation), $U(x, \alpha)$ halts within cT steps for some constant c depending only on N_α .*

1 Decision vs. Search

For much of this class, we focus on decision problems (languages), i.e., those with yes/no answers. Why do we do this when computation encompasses so many other types of problems?

1. This is the simplest setting for studying complexity, where we already can't answer most of the interesting questions we want to ask.
2. Solving decision problems actually lets us solve other types of computational problems!

Today, we'll address the second point by describing a generic "search-to-decision" reduction for NP. But to set up this general reduction, let's start by looking at a concrete problem.

Recall the decision problem

$$\text{SAT} = \{\varphi \mid \varphi \text{ has a satisfying assignment}\}.$$

This corresponds to a natural search problem:

SAT – Search: Given a CNF φ , output a satisfying assignment to φ if one exists, and $\perp$ otherwise.

Theorem 2. *If $\text{SAT} \in \mathbf{P}$, then there exists a poly-time algorithm solving SAT – Search.*

Proof. Let A be a poly-time algorithm deciding SAT. The idea is to repeatedly apply A to identify a satisfying assignment bit-by-bit.

First some notation: For a formula φ and $b \in \{0, 1\}$, let $\varphi|_{x_1=b}$ be the formula φ with variable x_1 replaced by the constant b . (And similarly if we want to fix more variables.)

Observe that:

$$\varphi \in \text{SAT} \iff \varphi|_{x_1=0} \in \text{SAT} \quad \text{or} \quad \varphi|_{x_1=1} \in \text{SAT}.$$

The following procedure uses A to construct a satisfying assignment $b_1, \dots, b_n$ to a formula $\varphi(x_1, \dots, x_n)$.

Alg Find(φ):

If $A(\varphi) = 0$: Output $\perp$.

For $i = 1, \dots, n$:

If $A(\varphi|_{x_1=b_1, \dots, x_{i-1}=b_{i-1}, x_i=0}) = 1$: // i.e., $\varphi|_{x_1=b_1, \dots, x_i=0}(x_{i+1}, \dots, x_n) \in \text{SAT}$

Set $b_i = 0$

Else:

Set $b_i = 1$

Output $(b_1, \dots, b_n)$

This algorithm uses $n + 1$ invocations of the poly-time algorithm A , so it runs in poly-time. $\square$

This result is just an example of a more general phenomenon. To state this, we need the following definition of an **NP** search problem.

Definition 3. Let $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$ be a poly-time computable function (“verification function”), and let $p : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomial (“certificate length”). The **NP** search problem associated to f and p is as follows.

Given $x \in \{0, 1\}^*$, find a $u \in \{0, 1\}^{p(|x|)}$ such that $f(x, u) = 1$ (if one exists).

For example, we can capture the SAT – Search problem by taking $f(\varphi, u) = 1 \iff u$ satisfies the CNF φ .

As another example, we can take $f(\langle G, k \rangle, u) = 1 \iff u$ is an independent set for G of size k .

Theorem 4. If $\mathbf{P} = \mathbf{NP}$, then every **NP** search problem can be solved in poly time.

Proof. Consider an **NP** search problem defined by verification function f and certificate length p . The idea will, again, be to construct a certificate bit-by-bit using our ability to solve a decision problem associated to f in poly-time.

One choice of decision problem that works here is

$$L_f = \{\langle x, b \rangle \mid \exists z \in \{0, 1\}^{p(|x|)-|b|} \text{ s.t. } f(x, b \circ z) = 1\}.$$

That is, L_f consists of pairs $\langle x, b \rangle$ such that there exists a way to extend the prefix b to a certificate $u = b \circ z$ for which $f(x, u) = 1$.

The language $L_f \in \mathbf{NP}$ because f is poly-time computable. Therefore, if $\mathbf{P} = \mathbf{NP}$, there is a poly-time algorithm A deciding L_f . This gives us the following search-to-decision reduction:

Alg Find(x):

If $A(\langle x, 0 \rangle) = 0$: Output $\perp$.

For $i = 1, \dots, p(|x|)$:

If $A(\langle x, b_1 \dots b_{i-1} 0 \rangle) = 1$: // i.e., $\langle x, b_1 \dots b_{i-1} 0 \rangle \in L_f$

Set $b_i = 0$

Else:

Set $b_i = 1$

Output $b_1 \dots b_{p(|x|)}$. $\square$

2 Hierarchy Theorems

Reductions and the theory of **NP**-completeness give us tools to understand the relative hardness of computational problems. But on their own, they don't give us a way to establish, say, that any of the problems in **NP** are themselves actually hard.

The main tool we have for actually separating complexity classes is diagonalization, which we'll first use here to prove hierarchy theorems. A hierarchy theorem says that strictly more of a particular resource lets you compute strictly more things. A few reminders about these resources:

- M runs in time $f(n)$ if it halts within $f(|x|)$ steps for every $x \in \{0, 1\}^*$.
- M runs in space $f(n)$ if it visits at most $f(|x|)$ cells of its work tapes for every $x \in \{0, 1\}^*$.

Definition 5. For a function $f(n)$, define the complexity classes

$$\mathbf{DTIME}(f(n)) = \{L \subseteq \{0, 1\}^* \mid L \text{ is decidable in time } O(f(n))\}.$$

$$\mathbf{SPACE}(f(n)) = \{L \subseteq \{0, 1\}^* \mid L \text{ is decidable in space } O(f(n))\}.$$

We'll start with the deterministic time hierarchy theorem, where the resource is time on a deterministic TM. To state the theorem precisely, we need a slightly annoying definition.

Definition 6. A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is time-constructible if there exists a TM computing $f(|x|)$ within $O(f(|x|))$ steps.

Most reasonable functions $f(n) \geq n$ are time-constructible, e.g., $n^{1.5}$, $n \log n$, 10^n . Some functions that are not time-constructible are $\log n$, $n^{0.9}$ (too small), Busy-Beaver(n) (too large to be computable).

Theorem 7 (Deterministic Time Hierarchy Theorem). *If f and g are time-constructible and $f(n) \log f(n) = o(g(n))$, then*

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n)).$$

That is, while $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DTIME}(g(n))$ (obviously), there exists a language $L \in \mathbf{DTIME}(g(n))$, but $L \notin \mathbf{DTIME}(f(n))$.

For example, if $f(n) = n$ and $g(n) = n \log^2 n$, then because $n \log n = o(n \log^2 n)$ we have that $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n \log^2 n)$.

Before proving the theorem, I want to draw attention to two apparent deficiencies. One is the “extra” factor of $\log f(n)$; ideally, one would like to be able to separate time classes whenever $f(n) = o(g(n))$. This factor has to do with the best known simulation of arbitrary-tape TMs by 3-tape TMs, which incurs the same logarithmic runtime blowup.

Second is the restriction to time-constructible f and g . This looks like a technicality, but it turns out to be important to the truth of the statement. There's a result called the Borodin-Trakhtenbrot Gap Theorem which implies there is a non-time-constructible function f for which $\mathbf{DTIME}(f(n)) = \mathbf{DTIME}(2^{f(n)})$.

As mentioned before, the idea we'll use to prove the time hierarchy theorem is diagonalization. This is the same idea that was implicit in our construction of the undecidable language SNA, so let's spell out that construction in a bit more detail.

Imagine constructing a 2-dimensional grid where rows are indexed by possible input strings x , and columns are indexed by Turing machines encoded by strings α . In each cell (x, α) , we write the value of $M_\alpha(x)$.

(Encodings of) Turing machines

Inputs		α_1	α_2	α_3	α_4	
	x_1	0 1	1	0	0	...
	x_2	1	1 0	0	000	
	x_3	11	10	1 1	0	
	x_4	1	0	1	100 1	
	$\vdots$					$\ddots$

The language SNA was defined by “flipping the diagonal” to force every TM M_α to make a mistake when run on input $x = \alpha$. That is,

$$\text{SNA} = \{x \mid M_x(x) \neq 1\}.$$

To prove our time hierarchy theorem, we’d like to use the same idea, but

1. The language we obtain by flipping the diagonal should be decidable in $O(g(n))$ time
2. We only need to thwart machines that run in time $O(f(n))$.

As a first attempt to do this, we might try only enumerating over all TMs running in time $O(f(n))$. The problem is that it’s undecidable to determine whether a given TM M runs in time $O(f(n))$. So instead, we’ll go back to enumerating over all TMs, but stop each simulation early, after only $g(n)$ steps.

Proof. We construct a language $L \in \mathbf{DTIME}(g(n)) \setminus \mathbf{DTIME}(f(n))$ as follows. Construct the TM $D(x) =$

1. Run the UTM $U(x, x)$ to simulate M_x on input x , and stop after the UTM has run for $g(|x|)$ time steps.
2. If U has output 1, return 0. Otherwise, return 1.

Let L be the language decided by D . By construction, $L \in \mathbf{DTIME}(g(n))$.

Now suppose for the sake of contradiction that some TM M decides L in time $O(f(n))$. By the efficiency of the UTM, simulating M to completion on any input x takes time $c_M f(|x|) \cdot \log f(|x|)$ for some c_M depending only on the machine M .

Now this is the part of the proof where some assumptions about how we encode TMs matter. Let us assume that every TM has infinitely many encodings (say, by appending an arbitrary number of trailing 0s). Then the fact that $f(n) \log f(n) = o(g(n))$ implies that there exists some encoding x^* of M such that $g(|x^*|) > c_M f(|x^*|) \cdot \log f(|x^*|)$.

Thus, when we run $D(x^*)$, the simulation has enough time to finish, and so $D(x^*) = 1 - U(x^*, x^*) = 1 - M(x^*)$, a contradiction. $\square$

The same proof goes through for space, but the hierarchy is even tighter because the UTM can simulate space-bounded computation with only a constant-factor overhead.

Theorem 8 (Deterministic Space Hierarchy Theorem). *If f and g are space-constructible and $f(n) = o(g(n))$, then*

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

3 Nondeterministic Time Hierarchy

A hierarchy theorem also holds for nondeterministic time. What do we have going for us when trying to prove such a theorem?

Good news: It's possible to simulate an arbitrary k -tape NTM by a 3-tape NTM using only a constant factor blowup in runtime. That is, there exists a universal NTM that can simulate any NTM running in time $T(n)$ using time $O(T(n))$.

Bad news: Central to our construction of the language L in the deterministic case was our ability to “flip” the answer produced by the UTM. It isn't so clear how to do this with the output of an NTM because of the asymmetry between its accept and reject conditions. (This is the same issue underlying the **NP** vs. **coNP** problem.)

Despite this fundamental issue, we can still prove a hierarchy theorem using a more clever diagonalization. Here's the statement:

Theorem 9. *Let f and g be time-constructible functions such that $f(n+1) = o(g(n))$. Then*

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

The proof in Arora-Barak is due to Zák, but I'll sketch a different (simpler, in my opinion) version due to Fortnow and Santhanam.

As before, we'll define the language in $\text{NTIME}(g(n)) \setminus \text{NTIME}(f(n))$ by an NTM D deciding it. This NTM will expect two inputs x, y . Let ε denote the empty string.

$$D(x, y) =$$

1. If $|y| < g(|x|)$: Use the UNTM to simulate $M_x(x, y0)$ and $M_x(x, y1)$, stopping each after the UNTM has run for $g(|x| + |y|)$ time steps. Accept iff both runs accept.
2. If $|y| \geq g(|x|)$: Use the UNTM to simulate $M_x(x, \varepsilon)$ using y as its choice of nondeterminism, stopping after it has run for $g(|x|)$ steps. Flip the answer.

Let L be the language decided by D . By construction, $L \in \text{NTIME}(g(n))$.

Now suppose for the sake of contradiction that some NTM N decides L in time $O(f(n))$. The fact that the UNTM runs with constant simulation overhead implies that it can simulate N in time $\leq cf(n)$ for some constant c (depending on N). Let x^* be an encoding of N such that $cf(|x^*| + m + 1) \leq g(|x^*| + m)$ for all $m \geq 0$, which exists because $f(n+1) = o(g(n))$. Then

$$(x^*, \varepsilon) \in L \iff N(x^*, 0) \text{ and } N(x^*, 1) \text{ both accept,}$$

since running the machine $D(x^*, \varepsilon)$ enters case 1, and $cf(|x^*| + 1) \leq g(|x^*|)$ implies that both simulations therein have time to run to completion. Since N decides L , this is equivalent to

$$(x^*, y) \in L \forall |y| = 1.$$

More generally, we have that for every $|y| < g(|x^*|)$,

$$(x^*, y) \in L \iff N(x^*, y0) \text{ and } N(x^*, y1) \text{ both accept.}$$

So we can repeat this argument, increasing the length of y one bit at a time, yielding

$$\begin{aligned} (x^*, \varepsilon) \in L &\iff (x^*, y) \in L \forall |y| = 1 \\ &\iff (x^*, y) \in L \forall |y| = 2 \\ &\dots \\ &\iff (x^*, y) \in L \forall |y| = g(|x^*|). \end{aligned}$$

Now for all strings y of length $g(|x^*|)$, running $D(x^*, y)$ enters case 2. So

$$(x^*, y) \in L \forall |y| = g(|x^*|) \iff N(x^*, \varepsilon) \text{ rejects for every choice of nondeterminism } |y| = g(|x^*|).$$

Since the (simulated) runtime of N is bounded by $cf(|x^*|) \leq g(|x^*|)$, this is equivalent to $N(x^*, \varepsilon)$ rejecting overall, which is a contradiction.