Lecture 3 - Diagonalization, time hierarchy.

Boaz Barak

February 13, 2006

Diagonalization While the general-purpose computer is a very useful and practical object, Turing also used this fact for negative results, following Godel in using the universality of a system to prove its limitations. This method is useful also in computational complexity, although its use seems to be more limited in this context (and in fact, there are even some proofs of its limitations).

Time hierarchy theorem One annoying possibility is that in fact there's not really such thing as the inherent complexity of problems. I mean by that for every problem there is an algorithm that solves it in time 2^n, a more complicated algorithm that solves it in time $2^{\sqrt{n}}$, an even more complicated $n^{\log n}$ algorithm, even more complicated n^{100} algorithm, and so on and so on until an immensely complicated $O(n)$-time algorithm (and maybe even here the constant can be improved). Note that the algorithms have to be more and more complicated since if there's an infinite sequence of improving algorithms then they must also grow in description size (i.e., their description size must also tend to infinity).[1] This turns out to actually be the case for some artificial problems with very unnatural hardness (this is called Blum's speed-up theorem). However, the question is whether this happens for all problems, and for example $\mathbf{DTIME}(n) = \mathbf{DTIME}(n^10)$ or even $\mathbf{DTIME}(n) = \mathbf{DTIME}(2^n)$.

While from the point of view of algorithms designers, this might be an attractive possibility since they always have room for improvement (although it kind of takes the fun and beauty out of algorithm design as one gets to algorithms with extremely long description full of tweaks and special cases). However, from the point of view of complexity theorists this is somewhat depressing, as it seems that there's not a lot of point to study the hardness of computational problems.

It turns out that this is not the case, and we can actually prove that for example $\mathbf{DTIME}(n^2) \neq \mathbf{DTIME}(n^4)$. Note that this is somewhat surprising, since we don't even know how to show that a problem like Hamiltonian cycle is not in $\mathbf{DTIME}(n)$ (even though we believe it's not in $\mathbf{DTIME}(2^{\sqrt{n}})$. Indeed, the hierarchy theorems (that you can do more stuff with more resources) are one of the very few cases where we can actually prove a separation between complexity classes.

Theorem 1. *For every $c \geq 1$, $\mathbf{DTIME}(n^c) \subsetneq \mathbf{DTIME}(n^{c+1})$.*

Proof. Define the following language L: $x \in L$ iff the x^{th} machine M_x outputs 0 on x within $|x|^{c+0.1}$ steps.

[1]This discussion demonstrates why I believe "complexity" is actually a bad name for what we're studying — for example CSAT is not a very complicated problem and we don't believe that it's optimal algorithm is very complicated. Rather we believe that the very simple brute force algorithm is roughly the best possible for it, and it's not complicated — just inefficient.

TBC ☐

Non-deterministic time hierarchy There exists a similar hierarchy theorem for *non-deterministic* time. Although previously we defined only the class **NP** it is clear that this definition can be generalized to $\mathbf{NTIME}(T)$ for any function $T : \mathbb{N} \to \mathbb{N}$:

Definition 1. Let $T : \mathbb{N} \to \mathbb{N}$ and let $L \subseteq \{0,1\}^*$. We say that $L \in \mathbf{NTIME}(T)$ if there exists a Turing machine M such that for every $x \in L$ and $y \in \{0,1\}^*$, $M(x,y)$ halts in $cT(|x|)$ steps for some constant $c > 0$ and

$$x \in L \iff \exists_{y\in\{0,1\}^*} \text{ s.t. } M(x,y) = 1$$

We have the following theorem:

Theorem 2. *For any* $c \geq 1$, $\mathbf{NTIME}(n^c) \subsetneq \mathbf{NTIME}(n^{c+1})$.

Proof. TBC ☐

$\mathbf{P}_{/\mathbf{poly}}$, non-uniformity We can use Boolean circuits as a different model of computation: If $f : \{0,1\}^* \to \{0,1\}^*$ is a function, we denote by f_n the restriction of f to $\{0,1\}^n$. We say that $f \in \mathbf{P}_{/\mathbf{poly}}$ if there are constants c, d such that for every n, f_n is computable by a circuit of size cn^d.

Every machine can be simulated by a circuit (in a similar way to the fact that CSAT is **NP**-complete) and so we have

Lemma 3. $\mathbf{P} \subseteq \mathbf{P}_{/\mathbf{poly}}$

Proof. Exercise ☐

Machines with advice Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function in $\mathbf{P}_{/\mathbf{poly}}$. This means that there is a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ where C_n computes $f_n = f_{\restriction\{0,1\}^n}$. Note that for every n, the circuit C_n can be completely different, and there need not be a uniform rule to obtain all these circuits. This is opposed to computing f by a Turing machine, in which case there's a single machine that computes f on all input length. For this reason Turing machines are often called a *uniform* model of computation, while circuits are called a *non-uniform* model of computation, and can in fact compute more functions. Thus, we have the following result:

Lemma 4. $\mathbf{P} \neq \mathbf{P}_{/\mathbf{poly}}$

We can also capture non-uniformity by Turing machines, if we allow them to take a string, often called *advice* for each input length.

Definition 2 (Computing with advice)**.** Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function, and let $T, a : \mathbb{N} \to \mathbb{N}$ be two functions. We say that $f \in \mathbf{DTIME}(T)_{/a}$ if there exists a Turing machine M, and a sequence of strings $\{\alpha_n\}_{n\in\mathbb{N}}$ with $|\alpha_n| \leq a(n)$ such that for every $x \in \{0,1\}^*$, if $n = |x|$ then $M(x, \alpha_n)$ outputs $f(x)$ within $T(n)$ steps.

I leave it to you to verify that the class $\mathbf{P}_{/\mathbf{poly}}$ is equal to the union of $\mathbf{DTIME}(n^c)_{/n^d}$ for all constants $c, d > 0$.

Note: As before, many texts define $\mathbf{P}_{/\mathbf{poly}}$ as a class containing only decision problems (i.e., functions with one-bit output).

$\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ implies PH collapses If $\mathbf{P} = \mathbf{NP}$ then we know that polynomial hierarchy collapses. If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then this does not mean that $\mathbf{P} = \mathbf{NP}$ since it may be hard to find that circuit. However, finding this circuit can be posed as a Σ_2-search problem, and once we have it, we can solve Π_2-problems in Σ_2.