Math 278 - Geometry and Algebra of Computational Complexity

Taught by David Hyeon
Notes by Dongryul Kim

Fall 2018

¡+instructor+¿ ¡+meetingtimes+¿ ¡+textbook+¿ ¡+enrolled+¿ ¡+grading+¿ ¡+courseassistants+¿

Contents

1 September 5, 2018

There are going to be biweekly homeworks, and a final writing project. The goal of the course is to introduce you to the various aspects of computational complexity theory. There will be four parts:

1. Turing machines, deterministic and non-deterministic, probabilistic algorithms, reduction, NP-completeness
2. Undecidable problems, Hilbert's 10th problem of solving diophantine equations
3. Computer models, continuous time systems, Blum–Smale–Shub model, quantum computers
4. Geometric complexity theory, algebro-geometric and representation theoretic approach to P≠NP

We may consider the determinant as a point in $\mathbb{P}(\mathrm{Sym}^n(\mathbb{C}^{n^2}))$. There is this conjecture that there is no constant $c \geq 1$ such that for all large m,

$$\mathrm{GL}_{m^{2c}}[\ell^{m^c - m}\,\mathrm{perm}_m] \notin \overline{\mathrm{GL}_{m^{2c}}[\det_{m^2}]}.$$

This implies P≠NP.

When you do any kind of programming at home, you use discrete time and discrete space. At the end, it really looks like

$$x_{k+1} = f(x_k).$$

On the other hand, the continuous time and space analogue will be a differential equation

$$y' = f(y).$$

Differential analyzers and continuous neural networks are like this. On the other hand, states in quantum computers lie in Hilbert spaces, and so they have continuous space but discrete time.

1.1 Turing machines

This is going to be boring. Let Σ be a finite set of alphabets, for instance, $\Sigma = \{0, 1\}$ for modern computers. Σ^* is the set of all words on Σ.

Definition 1.1. A **language** over Σ is a subset of Σ^*. A **decision problem** encoded on Σ is a partition

$$\Sigma^* = (\text{yes}) \amalg (\text{no}) \amalg (\text{non}).$$

(You get a yes or a no or an error.) The language associated to a decision problem Π is the "yes" part, and is denoted by L_Π.

Definition 1.2. A **deterministic Turing machine** has a read-write had, a bi-infinite tape, and a DTM program consisting of

- Σ a finite set of tape symbols, with $b \in \Sigma$ a blank symbol, and $\gamma \subseteq \Sigma$ a set of input symbols with $b \notin \gamma$,
- a finite set Q of states with distinguished q_0, q_Y, q_N of start, yes, no states,
- a transition function

$$\delta : (Q \setminus \{q_Y, q_N\}) \times \Sigma \to Q \times \Sigma \times \{\pm 1\}.$$

You should think of there being an infinite tape and a state-controller pointing to a certain point on the tape. The state-controller reads the tape symbol at that point, and plugs its own state and the tape symbol to δ. The output will be the new state of the state-controller, the symbol that will be written, and where the read-write head will move next. The program ends when either q_Y or q_N is hit.

On some inputs, a deterministic Turing machine may never halt. In fact, there is no "algorithm" that can determine whether a given deterministic Turing machine halts on a certain input. We will prove this shortly.

Example 1.3. Consider the following Turing machine. Find what this does.

$q \setminus \sigma$	0	1	b
0	$0, 0, 1$	$0, 1, 1$	$1, b, -1$
1	$2, b, -1$	$3, b, -1$	$N, b, -1$
2	$Y, b, -1$	$N, b, -1$	$N, b, -1$
3	$N, b, -1$	$N, b, -1$	$N, b, -1$

Definition 1.4. Let M be a deterministic Turing machine. The language recognized by M is

$$L_M = \{x \in \gamma^* : M \text{ accepts } x\}.$$

So M solves the decision problem Π if $L_M = \Pi$.

Definition 1.5. The **time complexity** of M is the function

$$T_M(n) = \max_{|x|=n}(m : M \text{ halts on } x \text{ in } m \text{ steps}),$$

where a step is a movement of the head.

2 September 10, 2018

Today we will talk about non-deterministic Turing machines.

2.1 Non-deterministic Turing machines

I will give two definitions, which are going to be equivalent. Recall that a deterministic Turing machine is just a infinite tape with a read-write head. The program really is the transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \to Q \times \Gamma \times \{\pm 1\}$. In a **non-deterministic Turing machine**, the picture is the same, but there are two transition functions δ_0 and δ_1. At each computational step, the machine makes an arbitrary choice between δ_0 and δ_1.

Definition 2.1. A **computation path** is the sequence of choices the machine makes. For instance, it looks like

$$\delta_0\delta_1\delta_0\delta_0\delta_1\delta_1 \cdots \text{ or } 010011\cdots .$$

The length of the computation path is going to be the length of the computation.

Definition 2.2. M is said to run in time $T(n)$ if for every input x and every computation path, the machine halts within $T(|x|)$ steps. We say that M is a **polynomial time** non-deterministic Turing machine if it runs in some polynomial time.

We say that M accepts x if there exists a computation path that halts with q_Y. Then we define the language accepted by M as

$$L_M = \{x \in \Sigma^* : M \text{ accepts } x\} \subseteq \Sigma^*.$$

Then we define

$$\mathcal{NP} = \{L \subseteq \Sigma^* : \text{exists a polynomial nDTM } M \text{ with } L_M = L\}.$$

It is clear that $\mathcal{P} = \mathcal{NP}$, because a DTM is always a nDTM. ($\mathcal{P}$ is the same thing with DTM instead of nDTM.) Intuitively, $\mathcal{NP}$ means that you can check an answer (computational path) in polynomial time.

Let me give an alternative definition of an nDTM. We now consider a two-tape machine, and we consider a transition function

$$\delta : Q \times \Gamma \times \Gamma \to Q \times \Gamma \times \Gamma \times \{\pm 1\} \times \{0, 1\}.$$

It also has a "guessing module". On an input x on the first tape, the guessing module writes an arbitrary guess y on second tape, of length bounded in polynomial by the length of x. Then the machine proceeds with the computation deterministically.

Definition 2.3. We say that M runs in time $T(n)$ if on an input x and for any guess, M halts in $T(|x|)$ steps.

Using this, we can again define $\mathcal{NP}$ so that L is in $\mathcal{NP}$ if there exists a language R (recognizable by a polynomial DTM) and a polynomial q such that

$$L = \{x : \exists y, |y| \leq q(|x|), (x, y) \in R\}.$$

In this case, we say that y is a "witness" or a "certificate" for x.

Theorem 2.4. *The two definitions are equivalent.*

Proof. Let L be $\mathcal{NP}$ according to the first definition. Then you can use the computation path as the guess. In particular, we can do something like

$$\delta(q, \sigma_1, \sigma_2) = (\sigma_2 \delta_1(\sigma_1, q) + (1 - \sigma_2)\delta_0(\sigma_1, q), 1).$$

The other direction does it similarly. $\square$

You can also define stuff like k-tape machines, but if you thing hard enough, you will see that there is no difference.

Definition 2.5. We say that a problem Π is **reduced** to Π' if there is a (polynomially) computable funciton

$$f : \Sigma^* \to \Sigma^*$$

such that $x \in L(\Pi)$ if and only if $f(x) \in L(\Pi')$.

What do we mean by a computable function? The easiest way to define it is by using a k-tape machine. This k-tape machine M has a dedicated input tape and an output tape. We say that M computes f if on input x, the content of the output tape is equal to $f(x)$ when the machine halts.

Definition 2.6. A problem or language is said to be **NP-hard** if any NP language can be polynomially reduced to it. It is said to be **NP-complete** if it itself is in NP.

If you search on Wikipedia, you can find hundreds of examples of NP-complete problems, mostly in discrete mathematics.

2.2 Encoding Turing machines

Now we want to encode a Turing machine, i.e., construct a map

$$\epsilon : \{0, 1\}^* \to \{\text{Turing machines}\}.$$

We are going to make a Turing machine on $\{0, 1, -\}$ and $Q = \{0, 1, 2, \ldots, l\}$. We encode ℓ and the transition function from values $\delta(\sigma, q)$ as a binary word. If any binary string does not come from this procedure, map it to some trivial Turing machine. This defines ϵ.

Definition 2.7. There exists a DTM $\mathcal{U}$ such that for every (x, α),

$$\mathcal{U}(x, \alpha) = M_\alpha(x).$$

This is called the **universal Turing machine**. If M_α halts on input x within T steps, then $\mathcal{U}$ halts in (x, α) within $CT \log T$ steps.

Our personal computers are all like this. If you write a program, you can run it. You can see at a high level how this will work. I was told that it is very involved to actually construct this machine.

3 September 12, 2018

3.1 Uncomputable functions

If you want to show that uncomputable functions exists, this is easy because there are countably many Turing machine, and uncountably many languages. So we want a construction of a function that is not computable by any DTM.

Example 3.1. Recall that we had this encoding of a DTM given by

$$\epsilon : \Sigma^* \to \{\text{DTMs}\}; \quad \alpha \mapsto M_\alpha.$$

Now define

$$f(\alpha) = \begin{cases} 0 & M_\alpha \text{ accepts } \alpha, \\ 1 & \text{else.} \end{cases}$$

Then we claim that f is not computable. Suppose that $M = M_{\alpha^*}$ computes f. Then

$$M_{\alpha^*}(\alpha^*) = 1 \quad \Leftrightarrow \quad f(\alpha^*) = 1 \quad \Leftrightarrow \quad M_{\alpha^*} \text{ does not accept } \alpha^*.$$

This is contradictory.

Example 3.2. Here is another example. Consider the problem of taking (α, x) and outputing whether M_α halts on input α. Suppose M_ξ solves the Halting problem HALT. We are then going to build a solution to the previous function by using the universal Turing machine. You first plug in (α, α) to M_ξ, and if it says no, just output 1. If it says yes, run $\mathcal{U}$ with α and α, and output the answer. This shows that the halting problem is undecidable.

Example 3.3. Let us look at the Bounded Halting Problem for nDTMs, denoted BHPN. First note that nDTMs can be encoded,

$$\epsilon : \Sigma^* \to \{\text{nDTMs}\},$$

and also that there is an efficient universal nDTMs. Now the input is (α, x, t), and the problem is,

> Does M_α halt on x on t steps?

This problem is $\mathcal{NP}$ because we can use the universal machine. On the other hand, it is $\mathcal{NP}$-hard as well. To see this, let $L \in \mathcal{NP}$ and let M be the nDTM that recognizes L. Then we can define

$$f : \Sigma^* \to \Sigma^*; \quad x \mapsto (\alpha, x, T(|x|)).$$

This reduces L to the Bounded Halting Problem. This shows that BHPN is $\mathcal{NP}$-complete.

Index