Topics in Theoretical Computer Science	March 14, 2016

Lecture 4 (Notes)

Lecturer: Ola Svensson | *Scribes: Ola Svensson*

Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works. They also contain extracts from the two main inspirations of this course:

1. The book *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak;
2. The course *http://theory.stanford.edu/ trevisan/cs254-14/index.html* by Luca Trevisan.

1 Introduction

Recall last lecture:

- Circuits: non-uniform computational model; the size of a circuit is the number of gates.
- A language L is in $\mathbf{SIZE}(T(n)))$ if there exists a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{Z}}$ such that for every $x \in \{0,1\}^n \Leftrightarrow C_n(x) = 1$.
- For every language L, $L \in \mathbf{SIZE}(O(2^n))$.
- At the same time almost all languages require circuits of size $\approx 2^n/n$. This followed from a counting argument.
- We let $\mathbf{P}_{/\mathbf{poly}} := \cup_c \mathbf{SIZE}(n^c)$.
- Kind of surprising at first, we showed that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages. The key reason is that it is a non-uniform computational model (we have a different circuit for each input length $n \in \mathbb{N}$).
- We also showed that $\mathbf{P} \subseteq \mathbf{P}_{/\mathbf{poly}}$ by showing that for any language $L \in P$ (and any $n \in \mathbb{N}$) we can in *polynomial time* construct a circuit C_n of polynomial size such that $C_n(x) = 1 \Leftrightarrow x \in L$ for all $x \in \{0,1\}^n$.

Today:

- We first give an alternative proof of the Cook-Levin Theorem using circuits.
- We then discuss randomized computing. What problems have better randomized algorithms than deterministic ones?
- Different randomized complexity classes.
- Connecting randomization to circuits (Adleman's Theorem).

2 Circuit Satisfiability and a proof of the Cook-Levin Theorem

Boolean circuits give an alternative proof of the central Cook-Levin Theorem that shows that 3-SAT is NP-complete.

Definition 1 (Circuit satisfiability or CKT-SAT) *The language CKT-SAT consists of all (strings representing) circuits that produce a single bit of output and that have a satisfying assignment.*

CKT-SAT is clearly in **NP** because the satisfying assignment can serve as the certificate. The Cook-Levin Theorem follows immediately from the next two lemmas.

Lemma 2 *CKT-SAT is* **NP***-hard.*

Proof

- If $L \in \mathbf{NP}$ then there is a polynomial-time TM M and a polynomial p such that $x \in L$ iff $M(x,u) = 1$ for some $u \in \{0,1\}^{p(|x|)}$.
- The proof that $\mathbf{P} \subseteq \mathbf{P_{/poly}}$ yields a polynomial-time transformation from M, x to a circuit C such that $M(x,u) = C(u)$ for every $u \in \{0,1\}^{p(|x|)}$. Thus $x \in L$ iff $C \in$ CKT-SAT.

■

Lemma 3 *CKT-SAT* $\leq_p$ 3-*SAT.*

Proof Map a circuit C into a 3-SAT formula φ as follows:

- For every node/gate v_i of C, we will have a corresponding variable z_i in φ.
- If the node v_i is an AND of the nodes v_j and v_k then we add to φ the clauses that are equivalent to the condition $z_i = (z_j \wedge z_k)$.
- Similarly, if v_i is an OR of v_j and v_k we add the clauses that are equivalent to $z_i = (z_j \vee z_k)$.
- And, if v_i is the NOT of v_j then we add the clauses that are equivalent to $z_i = \neg z_j$.
- Finally, if v_i is the output node of C then we add the clause (z_i) to φ.
- It is not hard to see that the formula φ is satisfiable iff the circuit C is. Moreover, the reduction runs in polynomial time.

■

3 Randomized computation

As this is a complexity course, we wish to understand the power of computing when we are allowed to flip a coin.

Let us first define randomized computation formally using probabilistic TMs.

Definition 4 *A* probabilistic Turing machine *(PTM) is a TM with two transition functions δ_0 and δ_1. To execute a PTM M on an input x, we choose in each step with probability* $1/2$ *to apply the transition function δ_0 and with probability* $1/2$ *to apply the transition function δ_1. The machine only outputs* 1 *("Accept") or* 0 *("Reject").*

This definition is a little abstract at first. It is indeed hard to design algorithms when thinking in this low level abstraction. The following less formal but intuitive definition can be helpful: A randomized algorithm is an algorithm that has the ability to toss coins.

3.1 Some examples of randomized algorithms

3.1.1 Finding a median

Given n integers $a_1, \ldots, a_n$ how do you find the median by a fast algorithm?

The standard way is to solve the following slightly more general problem: Given integers $a_1, \ldots, a_n$ and $1 \leq k \leq n$, find the k largest integer.

The randomized algorithm is recursive and works as follows

1. Pick a random $i \in [n]$ and let $x = a_i$.
2. Scan the list $\{a_1, \ldots, a_n\}$ and count the number m of a_i's such that $a_i \leq x$.
3. If $m = k$, then output x.
4. Otherwise, if $m > k$, then copy to a new list L all elements such that $a_i < x$ and find the k:th largest integer in L (which is a smaller instance).
5. Otherwise (if $m < k$) copy to a new list H all elements such that $a_i > x$ and find $k - m$:th largest integer in H (which again is a smaller instance).

An analysis similar to the analysis of QuickSort shows that this algorithm runs in expected linear time. There is also a deterministic algorithm that runs in linear time but it is much more involved and harder/slower to implement.

3.1.2 Polynomial identity testing

How do you efficiently check whether two polynomials P and Q are identical?

This is equivalent to checking whether a single polynomial is equal to zero, i.e., check whether $P - Q \equiv 0$.

We assume the polynomials are given implicitly (think determinant, permanent). Note that e.g. the polynomial $\prod_{i=1}^{n}(1 + x_i)$ can be evaluated efficiently but has 2^n many terms. This means that to check whether a polynomial is equivalent to 0 we can not afford to write out all the terms.

The simple randomized algorithm is based on the Schwartz-Zippel Lemma:

Lemma 5 *Let $p(x_1, \ldots, x_m)$ be a nonzero polynomial of total degree at most d. Let S be a finite set of integers. Then, if $a_1, \ldots, a_m$ are randomly chosen from S, then*

$$\Pr[p(a_1, a_2, \ldots, a_m) \neq 0] \geq 1 - \frac{d}{|S|}.$$

This suggest the following simple algorithm to check whether a polynomial p of degree d is equivalent to 0:

1. Choose $a_1, \ldots, a_m$ at random from $\{1, \ldots, 3d\}$.
2. Output that the polynomial is equivalent to 0 if $p(a_1, \ldots, a_m) = 0$.

Note that the algorithm is always correct if $p \equiv 0$. If $p \not\equiv 0$ then by Lemma 5 it succeeds with probability at least $2/3$. This probability can be boosted by repeating the algorithm more times.

It remains a major open problem to find an efficient deterministic algorithm for polynomial identity testing.

4 Two-sided, One-sided and Zero-Sided Error

In our examples, we saw different types of randomized algorithms. One that always reported a true answer with *expected* polynomial time running time. Another that always run in polynomial time but could with a small probability output the wrong answer. Let's make these differences formal.

Definition 6 (Two-sided error) *Let* $\mathbf{BPTIME}(T(n))$ *be the class of languages that contain language* L *if there is a probabilistic TM* M *running in time* $T(n)$ *satisfying*

$$\Pr[M(x) = L(x)] \geq 2/3 \qquad \textit{for every } x \in \{0,1\}^*.$$

Let $\mathbf{BPP} = \cup_c \mathbf{BPTIME}(n^c)$.

Definition 7 (One-sided error) $\mathbf{RTIME}(T(n))$ *contains every language* L *for which there is a probabilistic TM* M *running in* $T(n)$ *time such that*

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[M(x) = 0] = 1.$$

Let $\mathbf{RP} = \cup_c \mathbf{RTIME}(n^c)$.

We also define the class capturing the other one-sided error (on inputs not in the language) as $\mathbf{coRP} = \{L : \bar{L} \in \mathbf{RP}\}$.

- Note that polynomial identity testing is in **coRP**.

Definition 8 (Zero-sided error) *The class* $\mathbf{ZTIME}(T(n))$ *contains all the languages for which there is a machine* M *that runs in expected time* $O(T(n))$ *such that for every input* x*, whenever* M *halts on* x*, we have* $M(x) = L(x)$.

Define $\mathbf{ZPP} = \cup_c \mathbf{ZTIME}(n^c)$.

- Finding the median is morally in **ZPP**. (Only morally, since we didn't define it as a decision problem.)

5 Exercises

Exercise 1 *Show that* $\mathbf{RP} \subseteq \mathbf{NP}$.

Exercise 2 *Show that* $\mathbf{BPP} \subseteq \mathbf{EXP}$.

Embarrassingly, it is **not** known whether **BPP** is a strict subset of **NEXP** even though it is believed that $\mathbf{BPP} = \mathbf{P}$.

Exercise 3 *Let* $L \in \mathbf{BPP}$. *Show that there is a polynomial time probabilistic TM* M *such that*

$$\Pr[M(x) = L(x)] \geq 1 - \frac{1}{2^{|x|+1}}.$$

6 Error reduction: solution to Exercise 3

- You may have wondered why the constant 2/3 came up in the definition of **BPTIME** and **RTIME**.
- Well it is an arbitrary choice and it doesn't really matter because we can always improve our error probability by repetition.

To see that let us prove Exercise 3.

- As L is in **BPP**, there is a polynomial time probabilistic TM M' such that $\Pr[M(x) = L(x)] = 2/3$.
- The machine M simply does the following:

  For every input $x \in \{0,1\}^*$, run $M'(x)$ for $k = 100|x|$ times obtaining outputs $y_1, \ldots, y_k \in \{0,1\}$. If the majority of these outputs is 1, then output 1; otherwise, output 0.

- To analyze M, define for every $i \in [k]$ the random indicator variable X_i to equal 1 if $y_i = L(x)$ and to equal 0 otherwise.
- Note that $X_1, \ldots, X_k$ are *independent* Boolean random variables with $E[X_i] = \Pr[X_i = 1] = 2/3$. Note also that our algorithm returns the right answer if $X_1 + X_2 + \cdots + X_k > k/2$.
- Let $\mu = E[X_1 + \ldots X_k] = 2k/3$. Now applying a Chernoff bound yields that

$$\begin{aligned}\Pr[M \text{ outputs incorrect answer}] &\leq \Pr\left[|\sum_{i=1}^{k} X_i - \mu| \geq \frac{1}{4}\mu\right] \\ &< e^{-\Omega(\mu)} < 2^{-(n+1)},\end{aligned}$$

  by the choice of k.

- Hence, we have defined a polynomial time probabilistic TM M such that

$$\Pr[M(x) = L(x)] \geq 1 - \frac{1}{2^{n+1}} \qquad \text{for all } x \in \{0,1\}^*.$$

Notice that we can further improve the error probability by increasing the number of repetitions.

7 Adleman's Theorem: BPP $\subseteq$ P$_{/\mathbf{poly}}$

- As previously stated, it is believed that $\mathbf{P} = \mathbf{BPP}$.
- Therefore, we should expect that $\mathbf{BPP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ since $P \subseteq \mathbf{P}_{/\mathbf{poly}}$.

Theorem 9 (Adleman'78) $\mathbf{BPP} \subseteq \mathbf{P}_{/\mathbf{poly}}$.

Proof

- Let $L \in \mathbf{BPP}$. By Exercise 3, there exists a probabilistic TM M such that on inputs of length n satisfies

$$\Pr[M(x) = L(x)] \geq 1 - \frac{1}{2^{n+1}}.$$

- Here, the probability is over the random/probabilistic choices of M. If we let $M(x,r)$ denote the execution of M with random choices $r \in \{0,1\}^{poly(n)}$. Then we can write this probability as

$$\Pr_r[M(x,r) = L(x)] \geq 1 - \frac{1}{2^{n+1}} \qquad \text{, or equivalently as} \qquad \Pr_r[M(x,r) \neq L(x)] < \frac{1}{2^{|x|+1}}.$$

- Now consider all inputs $x \in \{0,1\}^n$ of length n. Then a simple union bound yields

$$\Pr_r[M(x,r) \neq L(x) \text{ for all } x \in \{0,1\}^n] \leq \sum_{x \in \{0,1\}^n} \Pr_r[M(x,r) \neq L(x)] < 2^n \cdot \frac{1}{2^{n+1}} = 1/2.$$

- This means that for each $n \in \mathbb{N}$, there exist random choices $r_n \in \{0,1\}^{poly(n)}$ such that

$$M(r_n, x) = L(x) \qquad \text{for all } x \in \{0,1\}^n.$$

- As $M(r_n, \cdot)$ is a deterministic polynomial time execution we can write down a polynomial size circuit C_n (in the same way we did last lecture) so that

$$C_n(x) = M(r_n, x) = L(x) \qquad \text{for all } x \in \{0,1\}^n.$$

- It follows that L has a polynomial sized circuit family $\{C_n\}_{n \in \mathbb{N}}$, which completes the proof.

■

8 Some comments

- Randomization seems to help when designing algorithms from an intuitive point of view.
- However, it is believed that it does not change the power of polynomial time computation, i.e., that $\mathbf{P} = \mathbf{BPP}$.
- It is actually open to prove that $\mathbf{BPTIME}(n) \subsetneq \mathbf{BPTIME}(n^{100})$ and $\mathbf{BPP} \subsetneq \mathbf{NEXP}$.
- Moreover, it is known that in order to prove $\mathbf{P} = \mathbf{BPP}$ one also has to prove new interesting circuit lower bounds.
- We are thus quite far from proving $\mathbf{P} = \mathbf{BPP}$ but at least we could show that $\mathbf{BPP}$ has polynomial size circuits.
- $\mathbf{P}_{/\mathbf{poly}}$ is a mysterious class. We know that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ would imply $\mathbf{P} \neq \mathbf{NP}$. However, is that reasonable to expect as $\mathbf{P}_{/\mathbf{poly}}$ also contains undecidable languages? In the next lecture, we will show that this is indeed reasonable to expect because otherwise something called the polynomial hierarchy collapses (Karp-Lipton Theorem).
- This gives the hope that we can resolve the $\mathbf{P}$ vs $\mathbf{NP}$ question by studying circuits. However, Razborov showed that such a proof would need to *not* be "natural".
- We can then continue to discuss circuit lower bounds in restricted models or start with interactive proofs. Any opinions?

9 Exercise

Exercise 4

1. *Prove that a language L is in* $\mathbf{ZPP}$ *iff there exists a polynomial-time PTM M with outputs in* $\{0,1,?\}$ *such that for every $x \in \{0,1\}^*$, with probability* 1, $M(x) \in \{L(x), ?\}$ *and* $\Pr[M(x) = ?] \leq 1/2$.
2. *Show that* $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.