Topics in Theoretical Computer Science April 4, 2016

Lecture 6 (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Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works.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:

- Polynomial hierarchy (alternation of quantifiers)
  - $L \in \Sigma_i^p$ if exists polytime TM M and polynomial q such that

    $$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \ldots, u_i) = 1 \qquad \forall x \in \{0,1\}^*,$$

  - $L \in \Pi_i^p$ if exists polytime TM M and polynomial q such that

    $$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \ldots, u_i) = 1 \qquad \forall x \in \{0,1\}^*,$$

- Karp-Lipton Theorem: If $\mathbf{NP} \subseteq \mathbf{P_{/poly}}$ then $\mathbf{PH} = \Sigma_2^p$.
- We also saw that $\mathbf{BPP} \subseteq \Sigma_2^p$, i.e., that two-sided randomized computing is in the second level of the polynomial hierarchy.

Today:

- Why does circuit lower bounds seem so difficult? One answer is that any proof showing $\mathbf{NP} \subsetneq \mathbf{P_{/poly}}$ cannot be a so called *natural proof*.
- Interactive proofs
  - **NP** can be defined as the class of languages which can be recognized by a polytime verifier given a proof/certificate (from a prover). This is an offline procedure.
  - What if the verifier and the prover can interact like we do in class?

2 Natural Proofs: a barrier for proving circuit lower bounds

There has been a large success in proving lower bounds in restricted models:

- Polysize circuits of unbounded fan-in for PARITY requires depth at least $O(\log n / \log \log n)$ (Håstad'86)
- CLIQUE does not have monotone circuits of polysize (Razborov'85)
- MATCHING does not have monotone circuits of polysize (Razborov'85)

(The latter result kind of destroyed the hope to use monotone circuits to separate **NP** from $\mathbf{P_{/poly}}$.)

In addition Karp-Lipton's Theorem tells us that separating **NP** from $\mathbf{P_{/poly}}$ seems like a good approach for the **P** vs **NP** problem.

However, our understanding of circuit lower bounds is embarrassing in general:

- Although almost all functions/languages require large circuits, we have been able to explicitly find any function that requires large circuits.
- In fact, it is open to find an explicit $f : \{0,1\}^n \to \{0,1\}$ that does not admit a $O(\log n)$ depth circuit of size $O(n)$.
- Another frontier is "Does **NEXP** have languages that require super-polynomial size circuits?". Or even "Does **NEXP** have languages that require $\omega(\log n)$ depth?".

In 1994, Razborov and Rudich described what they view as the main technical limitation of current approaches for proving circuit lower bounds:

- They defined the notion of "natural mathematical proofs" and pointed out that current lower bounds are (mostly) based on such proofs.
- They showed that obtaining strong lower bounds with such a proof would violate a stronger form of the $\mathbf{P} \neq \mathbf{NP}$ conjecture: namely that strong one-way functions do exist that cannot be inverted by algorithms running in subexponential time.
- Their result can be seen as a modern analog of the 1970s results on the limits of diagonalization (that we saw in class).

2.1 Definition of Natural Proofs

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and let $c \geq 1$ be some number. Any proof that f does not have a n^c-sized circuit can be viewed as exhibiting *some property* that f has, and which every function with n^c-sized circuits do not possess.

That is, such a proof can be seen as providing a predicate $\mathcal{P}$ on Boolean functions such that $\mathcal{P}(f) = 1$, but

$$\mathcal{P}(g) = 0 \qquad \text{for every } g \in \mathbf{SIZE}(n^c).$$

We call this condition n^c-usefulness as it rules out all the functions of circuits of size at most n^c.

So what are natural conditions on $\mathcal{P}$? Razborov and Rudich proposes two conditions:

Constructiveness: There is an $2^{O(n)}$-time algorithm that on input a function $g : \{0,1\}^n \to \{0,1\}$ outputs $\mathcal{P}(g)$. In other words, there is a polytime algorithm (in the size of the truth table of g) that evaluates $\mathcal{P}$.

Largeness: The probability that a random function $g : \{0,1\}^n \to \{0,1\}$ satisfies $\mathcal{P}(g) = 1$ is at least $1/n$.

Two examples of unnatural predicates:

1. $\mathcal{P}(g) = 1$ iff g is a Boolean function on n bits that has circuit complexity more than $n^{\log n}$. This predicate is clearly $c = o(n^{\log n})$ useful. Does $\mathcal{P}$ satisfy largeness? Yes almost all Boolean functions require exponential-sized circuits. However, we do not know if it satisfies constructiveness, since the trivial algorithm for computing it would be to enumerate all circuits of size $n^{\log n}$ which requires time $2^{n^{\log n}}$.
2. $\mathcal{P}(g) = 1$ iff g correctly solves the decision problem 3SAT on inputs of length n. This function is constructive: to compute it just enumerate all size n instances of 3SAT and check that g is correct on all instances. This takes $2^{O(n)}$ time. However, $\mathcal{P}$ does not satisfy the largeness property as it is 1 for exactly one function.

2.2 What's "Natural" about Natural Proofs?

The intuition of the constructiveness criteria is that most proofs use constructive properties. History tells us that it often is hard to exploit properties that do not have efficient algorithms. In fact many unconstructive arguments such as Lovasz Local Lemma and discrepancy was later constructivized. That said, there are of course unconstructive proof techniques (e.g. probabilistic method).

So what about largeness? Why should a lower bound for a specific function use a property that holds with good probability for a random function as well? The intuition is as follows

- Actually any proof that a function $f_0\{0,1\}^n \to \{0,1\}$ does not have a size S circuit, implies that at least half of the functions from $\{0,1\}^n$ to $\{0,1\}$ do not have a circuit of size $S/2 - 10$.
- To see this, choose a random $g : \{0,1\}^n \to \{0,1\}$ and observe that

$$f_0 = (f_0 \oplus g) \oplus g,$$

  where $g \oplus h$ denotes the function that maps every input x to $g(x) \oplus h(x)$.
- Then we can see that if both $(f_0 \oplus g)$ and g have circuits of size $< S/2 - 10$ then f_0 has a circuit of size $< S$.
- Since both g and $(f_0 \oplus g)$ are uniformly distributed, it follows that at least half of the functions must have circuits of size at least $S/2 - 10$.

2.3 Statement and explanation of result

Theorem 1 *Suppose that subexponentially strong one-way functions exist. Then there exists a constant $c \in \mathbb{N}$ such that the is no n^c-useful natural predicate.*

The assumption of the theorem is basically as follows: There are polytime computable functions $f : \{0,1\}^* \to \{0,1\}$ such that any algorithm, that given y (with good probability) finds an x such that $f(x) = y$ needs time 2^{n^ϵ} for some $\epsilon > 0$. We believe that such functions exist (think crypt e.g. factoring).

- Note that theorem implies that if subexponentially strong one-way functions exist, then no natural proof can prove $\mathbf{NP} \subsetneq \mathbf{P}_{\mathbf{/poly}}$.

We do not cover the proof in this part of the course but it is given with (much) more interesting information in Chapter 23 of the textbook.

3 Interactive Proofs

- Standard notion of a mathematical proof is closely related to the certificate definition of **NP**:

  > To prove that a statement is true one provides a sequence of steps/symbols, and the verifier checks that they present a valid proof/certificate.

- However, people often use a more general way to convince one another of the validity of statements:

  > They *interact* with one another, where the person verifying the proof (called *verifier*) asks the person providing it (the *prover*) for a series of explanations before he is convinced.

- Interactive proofs aim to understand such proofs from a complexity-theoretic perspective.
- We will see that if the verifier has to be deterministic then we can still only recognize **NP**, i.e., interaction does not add any power.

- However, if we add randomness to the verifier, then we get **PSPACE**!
- When adding randomness to the verifier one can distinguish between *private* vs *public* coins. We will discuss this difference in this lecture and prove the **PSPACE** result in the next lecture.

3.1 Warm up: Interactive Proofs with deterministic verifier (and prover)

When defining interactive proofs, it makes sense to allow the prover any computational power (after all, he or she can have proved something really great) but it should be easy to verify. Therefore, we will always require the verifier to run in polynomial time (but have no restrictions on the prover)!

Let us now formally define "interaction".

Definition 2 *Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be functions and k an integer (allowed to depend on input size). A k-round interaction of f and g on input $x \in \{0,1\}^*$, denoted by $\langle f, g\rangle(x)$ is the sequence of strings $a_1, \ldots, a_k$ defined as follows:*

$a_1 = f(x)$ *(verifier computes first question based on input)*

$a_2 = g(x, a_1)$ *(prover answers first question)*

$a_3 = f(x, a_1, a_2)$ *(verifier computes first question based on input and previous answers)*

$a_4 = g(x, a_1, a_2, a_3)$ *(prover answers second question)*

$\vdots$

$a_{2i+1} = f(x, a_1, \ldots, a_{2i})$ *for* $2i < k$

$a_{2i+2} = g(x, a_1, \ldots, a_{2i+1})$ *for* $2i + 1 < k$

The output *of f at the end of the interaction denoted $out_f\langle f, g\rangle(x)$ is defined to be $f(x, a_1, \ldots, a_k)$; we assume this output is in $\{0,1\}$.*

Having defined interaction, the definition of a deterministic proof system follows quite naturally.

Definition 3 *We say that a language L has a k-*round deterministic proof system *if there is a deterministic TM V that on input $x, a_1, \ldots, a_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any function P such that*

(Completeness) $x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^* : out_V\langle V, P\rangle(x) = 1$.

(Soundness) $x \notin L \Rightarrow \forall P : \{0,1\}^* \to \{0,1\}^* : out_V\langle V, P\rangle(x) = 0$.

The class **dIP** *contains all languages with a $k(n)$-round deterministic interactive proof system where $k(n)$ is polynomial in $n = |x|$.*

3.2 Exercises

Exercise 1 *Show that* $\mathbf{dIP} = \mathbf{NP}$.

Exercise 2 *Give a randomized interactive proof for the following problem:*

- *Marla (prover) has one red sock and one yellow sock, but her friend Arthur (verifier), who is color-blind, does not believe her that the socks have different colors. How can she convince him that this is really the case?*

We assume that Arthur takes the two socks from Marla, and he can then flip coins and then ask Marla questions based on the outcome of these random tosses. How can he do it so as to get convinced (with reasonable probability) that Marla has two different socks if it is the case?

Exercise 3 *This exercise is about the largeness condition of natural proofs. Prove that a random function* $g : \{0,1\}^n \to \{0,1\}$ *satisfies* $\mathcal{P}(g) = 1$ *with high probability, where* $\mathcal{P}$ *is the property that for no fixing of* $n - n^\epsilon$ g*'s inputs turns* g *into a constant function.*

3.3 The class IP: Probabilistic Verifier

We want to capture the additional power of a probabilistic verifier seen in Exercise 2. To this end, we extend "deterministic" interaction to that of including a probabilistic verifier:

- The verifier f is now probabilistic, so we add an additional m-bit input r to the function f.
- That is the interaction is now

$$a_1 = f(x, r)$$
$$a_2 = g(x, a_1)$$
$$a_3 = f(x, r, a_1, a_2)$$

  and so on.

- Notice that the prover (g) does *not* see the random coins. For this reason this is called *private coins* as opposed to *public coins* that we discuss later.
- The interaction $\langle f, g\rangle(x)$ is now a random variable of $r \in \{0,1\}^m$. Similarly the output $\text{out}_f\langle f, g\rangle(x)$ is also a random variable of r.

Definition 4 (Probabilistic verifiers and the class IP) *For an integer $k \geq 1$, we say that a language L is in $\mathbf{IP}[k]$ if there is a probabilistic TM V that on input $x, r, a_1, \ldots, a_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any function P such that*

(Completeness) $x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^* : \Pr[out_V\langle V, P\rangle(x) = 1] \geq 2/3$.

(Soundness) $x \notin L \Rightarrow \forall P : \{0,1\}^* \to \{0,1\}^* : \Pr[out_V\langle V, P\rangle(x) = 1] \leq 1/3$.

We define $\mathbf{IP} = \cup_{c\geq 1}\mathbf{IP}[n^c]$.

Remark We give the following remarks:

- The constants $2/3$ and $1/3$ are arbitrary and they can be made very small (just as for **BPP**) using sequential repetition that repeats the basic protocol m times and takes the majority answer. In fact, using a more complicated proof, it can be shown that we can decrease the probability of error without increasing the number of rounds using something called parallel repetition (where we run the m executions in of the protocol in parallel).
- Replacing the constant $2/3$ with 1 in the completeness requirement does not change the class **IP**. (This is a non-trivial fact.)
- In contrast, replacing the constant $1/3$ with 0 in the soundness case is equivalent to having a deterministic verifier and hence reduces the class **IP** to **NP**.

3.3.1 Interactive proof for graph nonisomorphism

We give an example of a language in **IP** that is not known to be in **NP**.

- Two graphs G_1 and G_2 are *isomorphic* if they are the same up to a renumbering of vertices; in other words, if there is a permutation π such that $\pi(G_1) = G_2$.
- The GI problem is the following: given G_1, G_2 decide if they are isomorphic.
- Clearly GI is in **NP**. Why? It is actually not known whether GI is **NP**-complete or in **P**. Together with factoring, it is the most famous such unresolved problem. Recently (this year), Laszlo Babai made major progress by showing that GI can be solved in time $O(n^{\log n})$.

Here we give an interactive proof for the complementary problem GNI that is not known to be in **NP**. That is GNI is the following problem: decide whether two given graphs G_1, G_2 are *not* isomorphic.

Protocol: Private-coin Graph Nonisomorphism (The protocol is very similar to the "sock"-protocol.)

V: Pick $i \in \{1, 2\}$ uniformly at random. Randomly permute the vertices of G_i to get a new graph H. Send H to P.

P: Identify which of G_1, G_2 was used to produce H. Let G_j be that graph. Send j to V.

V: Accept if $i = j$; reject otherwise.

As in the "sock"-protocol, it is easy to see that if the graphs are not isomorphic then the verifier always accepts. However, if they are isomorphic, the verifier accepts with probability $\leq 1/2$ (repeating the protocol decreases this probability further).

Remark One can see that in the above protocol the prover reveals no information except that the graphs are nonisomorphic. This is called a zero-knowledge proof because the verifier learns nothing except the truth of the statement. This has been formalized and studied. It has wide spread applications. For example, to prove that you hold the password without revealing the password itself.

3.4 Public Coins and AM

Our proof system for the "socks" problem and graph nonisomorphism seemed to crucially rely on the verifier's access to a source of *private* random coins that are not seen by the prover. Allowing the prover full access to the verifier's random string leads to the model of *interactive proofs with public coins*

Definition 5 (AM) *For every k, the complexity class $\mathbf{AM}[k]$ is defined as the subset of $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be random bits and not allowing it to use any other random bits that are not contained in these messages.*

- An interactive proof where the verifier has this form is called a *public coin* proof, sometimes also known as an Arthur-Merlin proof.
- Note that we are not assuming that the verifier sends any other messages than the random coins. This is w.l.o.g. as the almighty prover can reconstruct everything by seeing the random coins of the verifier.
- Also note that the verifier does not get to see all random coins at once but rather they are revealed to the prover iteratively message by message.
- We denote (slightly inconsistently) by $\mathbf{AM}$ the class $\mathbf{AM}[2]$. That is $\mathbf{AM}$ is the class of languages with an interactive proof that consists of the verifier sending a random string, and the prover responding with a message, where the verifier's decision is obtained by applying a deterministic polynomial-time function to the transcript.

3.4.1 Simulating Private Coins by Public Coins

Clearly, for every k, $\mathbf{AM}[k] \subseteq \mathbf{IP}[k]$. More surprisingly:

Theorem 6 (Goldwasser-Sipser'87) *For every $k : \mathbb{N} \to \mathbb{N}$ with $k(n)$ computable in poly(n),*

$$\mathbf{IP}[k] \subseteq \mathbf{AM}[k + 2].$$

Instead of proving this theorem in its full generality we prove the also surprising fact:

Theorem 7 GNI $\in$ **AM**[2]

The proof of this theorem demonstrates the power of recasting the problem. Indeed, it seems very hard to simply adapt the protocol with private coins that we saw previously. Instead, look at graph nonisomorphism in the following more quantitative way:

- Consider the following set of labeled graphs

  $$S = \{H : H \text{ is isomorphic to } G_1 \text{ or } G_2\}.$$

  Note that it is easy to certify that a graph H is a member of S by providing the permutation mapping of either G_1 or G_2 to H.

- An n-vertex graph has at most $n!$ equivalent graphs (all permutations). For simplicity, assume that both G_1 and G_2 have each exactly $n!$ equivalent graphs (symmetries can be dealt with appropriately).

- Then, we have

  - If G_1 is not isomorphic to G_2, $|S| = 2n!$ (and $|S \times S \times S \times S| = 16(n!)^4$).
  - If G_1 is isomorphic to G_2, $|S| = n!$ (and $|S \times S \times S \times S| = (n!)^4$).

- Hence, if we let $S' = S \times S \times S \times S$, we have reduced the problem of the prover to that of convincing the verifier that $|S'| \geq 16(n!)^4$ holds instead of $|S'| \leq (n!)^4$.

- This is done by a *set lower bound protocol* that we know describe.

Set Lower Bound Protocol (Simplified version of Goldwasser-Sipser protocol):

Conditions: S is a set such that a membership in S can be certified. Both partities know a number K. The prover's goal is to convince the verifier that $|S| \geq K$ and the verifier should reject with good probability if $|S| \leq K/16$.

V: Randomly pick a hash function $h : S \to \{1, 2, \ldots, K/4\}$ (that sends each element in S to a random element in $\{1, \ldots, K/4\}$. Pick $y \in \{1, \ldots, K/4\}$ uniformly at random. Send h, y (or equivalently the randomly used bits) to the prover.

P: Try to find an $x \in S$ such that $h(x) = y$. Send such an x to V together with a certificate that $x \in S$.

V's output: If $h(x) = y$ and the certificate validates that $x \in S$ then accept; otherwise reject.

Remark The hash function as written now would take too many random bits to encode, namely $|S| \log(K/4)$. However, one can prove that a family of pairwise independent hash function suffices (not too hard, see textbook). Such a function can then be sent using $O(\log |S|)$ random bits which is polynomial.

Let us now analyze the protocol. Clearly, the prover (being all powerful) can make the verifier accept iff h, y happen to be such that $x \in S$ exists satisfying $h(x) = y$.

Claim 8 *The protocol has completeness at least* $2/3$*, i.e., if* $|S| \geq K$ *then the verifier outputs* 1 *with probability at least* $1/2$*.*

Proof

- Let y be the randomly chosen element in $\{1, \ldots, K/4\}$.
- The probability that h hashes an element to from S to x is

$$1 - \Pr[h(x) \neq y \text{ for all } x \in S] = 1 - \left(1 - \frac{4}{K}\right)^K \geq 2/3.$$

(For the equality we used that the hash function sends each element in S to a random element in $\{1, \ldots, K/4\}$ independently and uniformly at random.)

■

Claim 9 *The protocol has soundness at most* $1/3$*, i.e., if* $|S| \leq K/16$*, then the verifier outputs* 1 *with probability at most* $1/2$*.*

Proof

- Let y be the randomly chosen element in $\{1, \ldots, K/4\}$.
- The elements in S are at most hashed to $K/16$ elements in $\{1, \ldots, K/4\}$, i.e., a $1/4$ fraction of the elements.
- Since the hashing is uniform, the probability that there exists $x \in S$ such that $h(x) = y$ is at most $1/4$.

■

This finishes the analysis of the set lower bound protocol and thereby also of the fact that GNI $\in$ **AM**.

Some remarks:

- Theorem 6 is a bit harder but similar: in that case the idea is to make the set S contain all the (private) random strings that makes the private-coin verifier accept.
- Our protocol for set lower bound does not have perfect completeness. However, this can be achieved and thus implying a perfectly complete public-coins proof for GNI.
- Moreover, (as almost always) the soundness can be made exponentially small in the size of the input.