Topics in Theoretical Computer Science April 18, 2016

Lecture 8 (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Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works.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:

- Graph Ismorphism is unlikely to be **NP**-complete. Indeed if GI is **NP**-complete then the polynomial hierarchy collapses to its second level.
- **IP = PSPACE**
  - Arithmetization: define a polynomial that is 1 for satisfying assignments and 0 for unsatisfying assignments.
  - Sumcheck protocol: recursively check whether the polynomial equals a value K.
- One can also consider interactive proofs with multiple provers.
  - The power of two provers is equivalent to that of any polynomial number of provers.
  - If we let **MIP** be the class of languages with interactive proofs consisting polynomial interaction with polynomially many (two) provers, then **MIP = NEXP**. So many provers seem to add power to the verifier (think of interrogation: more informative if the two criminals cannot talk to each other).

Today:

- Another view of proofs: probabilistic checkable proofs.
- How can we verify a proof by only reading a constant number of bits?
- Important connection to the theory of approximability of **NP**-hard problems.

2 Probabilistically Checkable Proofs(PCP)

In this section we will discuss probabilistically checkable proofs, an idea having many implications in the development of modern theoretical computer science. Before we jump into PCPs, let us first recall the "verifier" definition of **NP**.

2.1 Preliminaries

NP: A language L is in NP iff there exists a polynomial time verifier V which when given an x and a proof Π of length $poly(|x|)$ verifies whether $x \in L$, with the following properties:

- **completeness:** If $x \in L$ then $\exists$ proof Π such that $V^{\Pi}(x)$ accepts.
- **Soundness:** If $x \notin L$ then $\forall$ proofs Π, the verifier $V^{\Pi}(x)$ rejects.

In the definition of **NP** the deterministic verifier reads the entire proof and the completeness and soundness hold with probability one.

Now an interesting question to ask is, what if we make our verifier to be non-deterministic and are willing to tolerate some non-zero probability of accepting a false proof, can we have our verifier read few bits of the proof and verify with high probability?.

Surprisingly, it is possible to have the verifier read a constant number of bits from the proof and verifies with high probability.

2.2 Definition of Probabilistically Checkable Proofs (PCPs)

PCP: A language L is in PCP(r, q) if there exists a probabilistic polynomial time verifier V which, when given an instance x and a proof Π of length $poly(|x|)$, proceeds as follows

1. Reads x and $O(r)$ random bits.
2. Based on x and $O(r)$ random bits, queries $O(q)$ bits from Π.
3. Computes and either accepts or rejects with following properties,
   - **completeness:** If $x \in L$ then $\exists$ proof Π such that $V^{\Pi}(x)$ accepts with probability 1.
   - **Soundness:** If $x \notin L$ the $\forall$ proof Π, the verifier $V^{\Pi}(x)$ accepts with probability at most $\frac{1}{2}$.

Theorem 1 (PCP Theorem (AS-ALMSS'92)) $\mathbf{NP} = \mathbf{PCP}(\log(n), 1)$.

We will not prove the PCP theorem in class (a couple of year's ago we did but it took 4 lectures), instead we will prove a weaker (but probably still surprising) result: $\mathbf{NP} = \mathbf{PCP}(poly(n), 1)$.

2.3 Exercises

Exercise 1 *Prove that* $\mathbf{PCP}(\log(n), 1) \subseteq \mathbf{NP}$.

Proof Consider a language L $\in$ PCP(log(n), 1), which by definition has a probabilistic polynomial time verifier. The claim is that from this probabilistic verifier we can construct a deterministic verifier. Given a proof Π and an input instance x, the deterministic verifier runs the probabilistic verifier for all combinations of the $clog(n)$ random bits, a total of $2^{clog(n)} = n^c$ runs, and accepts iff the PCP verifier accepts for every bit strings. Since the probabilistic verifier runs in polynomial time and the number of possible combinations is bounded by n^c the deterministic verifier runs in polynomial time. Completeness follows directly from the completeness of the probabilistic verifier. Soundness follows as for at least half of the bit strings we have a rejection in the probabilistic verifier, and thus a rejection by the deterministic verifier. In fact it suffices to use only $n^c/2 + 1$ of the random bitstrings, as at least one of them will reject. ■

Exercise 2 *Given that GAP 3-SAT is NP-complete, prove that* $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$.

GAP 3-SAT instances are 3-SAT instances with the constraint that either they are satisfiable or no more than $1 - \epsilon$ of the clauses can be satisfied simultaneously. Deciding whether a GAP 3-SAT instance is satisfiable is in some sense easier than deciding if a general 3-SAT instance is satisfiable due to the ϵ-gap separating the satisfiable instances from the non-satisfiable ones.

Remark It can be proven that the PCP-Theorem is equivalent to that of GAP 3-SAT being NP-complete. **Proof** We will prove that GAP 3-SAT $\in$ PCP($\log n$, 1). Then, as any problem in NP can be reduced in polytime to GAP 3-SAT (GAP 3-SAT is NP-complete by assumption), we can construct a probabilistic verifier for any problem in NP: First reduce it to GAP 3-SAT and use the verifier for that problem (which we will now present).

We prove that GAP 3-SAT $\in$ PCP($\log n$, 1). First consider reading $\log n$ random bits, and choosing one of the n clauses accordingly (let the bits read be treated as the binary representation of the clause). Read the 3 assignments in the proof which the clause uses. If the proof is correct, the clause is satisfied (correctness). If the proof is incorrect there is a probability at most $1 - \epsilon$ of accepting, since with probability at least ϵ the randomly selected clause will be not satisfied. If $\epsilon < 0.5$ this is not good enough. To drive this probability down, we read $\lceil \frac{1}{\epsilon} \rceil \log n$ random bits from which we choose $\lceil \frac{1}{\epsilon} \rceil$ clauses. The resulting false acceptance rate is then $(1-\epsilon)^{\lceil \frac{1}{\epsilon} \rceil}$, which is strictly less than $\frac{1}{2}$. Thus GAP 3-SAT is in PCP($\log n$, 1) with constants $C_r = \lceil \frac{1}{\epsilon} \rceil$ and $C_q = 3\lceil \frac{1}{\epsilon} \rceil$.

■

3 Preliminaries

In order to prove **NP** $\subseteq$ **PCP**($poly(n), 1$), we need to devise a probabilistic verifier for an **NP**-complete problem (namely QUADEQ), that runs in polynomial time, performs $O(poly(n))$ random coin flips, and reads $O(1)$ bits from the proof. In this lecture, we will first introduce the Walsh-Hadamard code (WH), then use its properties to design the required verifier.

4 Walsh-Hadamard code

Definition 2 *: The WH code encodes n bits into 2^n bits. The encoding function WH: $\{0,1\}^n \to \{0,1\}^{2^n}$ maps the input $u \in \{0,1\}^n$ into the truth table of the function $x \odot u = \sum_{i=1}^{n} x_i u_i$ mod 2.*

Let's see an example where we encode all possible bit strings x of length 2 into 4 bits :

- For WH(0,0), $x \mapsto (0,0) \odot x$.
- For WH(0,1), $x \mapsto (0,1) \odot x = x_2$.
- For WH(1,0), $x \mapsto (1,0) \odot x = x_1$.
- For WH(1,1), $x \mapsto (1,1) \odot x = x_1 \oplus x_2$.

x	WH(0,0)	WH(0,1)	WH(1,0)	WH(1,1)
(0,0)	0	0	0	0
(0,1)	0	1	0	1
(1,0)	0	0	1	1
(1,1)	1	1	1	0

We can see from this example that the codewords of Walsh-Hadamard code are the truth table of all linear functions over $\mathbb{F}_2^n$. We will see in the following subsection that the WH code has two interesting properties, namely its *local testability* and *local decodability.*

4.1 Local testability of the Walsh-Hadamard

Given a function $f : \{0,1\}^n \mapsto \{0,1\}$, we would like to whether there exist $u \in \{0,1\}^n$ such that $f = WH(u)$. In other word words, we want to know if the function f is linear. f is linear if and only if $f(x+y) = f(x) + f(y)\ \forall x, y \in \{0,1\}^n \bmod 2$. In order to check if f is linear, we do the following test.

Linearity test

1. Select $x, y \in \{0,1\}^n$ independently uniformly at random.
2. Check if $f(x+y) = f(x) + f(y) \bmod 2$.

We see that all linear functions are always accepted.

Definition 3 *Two functions f, g are $(1-\epsilon)$-close if they agree on all, but ϵ-fraction of the inputs.*

Theorem 4 *:* **The Blum, Luby, Rubinfeld (BLR) linearity test**
Let $f : \{0,1\}^n \mapsto \{0,1\}$ if $Pr(f(x+y) = f(x) + f(y)) \geq \rho$ for $\rho \geq \frac{1}{2}$ then f is ρ close to some linear function.

Corollary 5 *:*
For any $\delta \in (0, \frac{1}{2})$, there exists a linearity test that reads $O(\frac{1}{\delta})$ bits such that:

- *(Completeness:) if f is linear, the test accepts with probability 1.*
- *(Soundness:) if f is not $(1-\delta)$-close to a linear function, then the test accepts with probability $p \leq \frac{1}{2}$.*

Proof We repeat the test of the theorem $\frac{1}{\delta}$ many times independently.

- Completeness: f is linear then we always accepts
- Soundness: if f is not $(1-\delta)$-close to a linear function then a single test accepts with probability $p \leq 1-\delta$. Since we do $\frac{1}{\delta}$ tests independently, the probability that they all succeed is $\leq (1-\delta)^{\frac{1}{\delta}} \leq \frac{1}{e}$.

■

4.2 Local decodability of the Walsh-Hadamard code

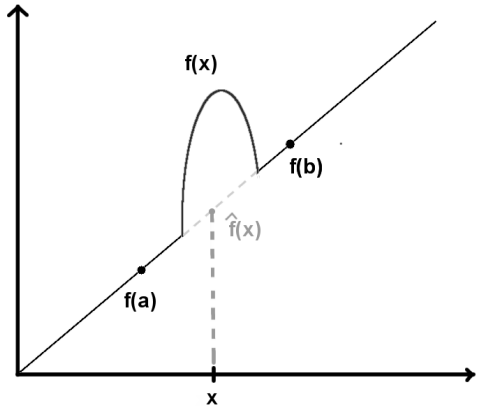
Given a function f that is $(1-\delta)$-close to some linear function $\hat{f}$, how to find $\hat{f}(x)$?
Since the function $\hat{f}(x)$ is linear we can simply compute $f(x) = \frac{1}{2} \cdot (f(a) + f(b))$

1. Select x'$\in \{0,1\}^n$ uniformly at random.
2. Output $f(x+x') + f(x')$.

Claim 6 *The algorithm outputs $\hat{f}(x)$ with probability $\geq 1 - 2 \cdot \delta$*

Proof Since f is $(1-\delta)$-close to $\hat{f}$, we have that

$$\Pr\left[f(x+x') = \hat{f}(x+x')\right] \geq 1-\delta$$
$$\Pr\left[f(x') = \hat{f}(x')\right] \geq 1-\delta$$



Hence by the union bound we get:

$$\Pr\left[f(x+x') = \hat{f}(x+x') \wedge f(x') = \hat{f}(x')\right] \geq 1 - 2\delta$$

In that case we get:

$$f(x+x') + f(x') = \hat{f}(x') = \hat{f}(x+x') + \hat{f}(x') = \hat{f}(x) + \hat{f}(x') = \hat{f}(x)$$

■

We will prove that NP $\subseteq$ PCP(poly(n),1) by showing that QUADEQ, an NP-complete problem, has a probabilistic verifier that uses $O(poly(n))$ random bits and queries $O(1)$ bits from proof.

5 The QUADEQ Problem

In order to prove that $NP \subset PCP(poly(n), 1)$, we will prove that an NP-complete problem has a proof-verifier that uses $O(Poly(n))$ random bits and queries $O(1)$ bits from the proof.

Definition 7 *(QUADEQ) Given a system of m quadratic equations over n variables in $\mathbb{F}_2$, decide whether there exists an assignment of the variables such that all equations are satisfied.*

Example 1 *Given the following system:*

$$u_1u_2 + u_3u_1 + u_2u_3 = 1$$
$$u_2u_2 + u_1 = 0$$
$$u_3u_2 + u_2u_1 + u_1u_1 = 1$$

The system is satisfied when $u_1 = u_2 = u_3 = 1$.

Since we are working in $\mathbb{F}_2$, we have that $x = x^2$, hence by replacing any linear term by its square value, we will only have quadratic terms in our equations. For n variables, there are n^2 different possible quadratic terms (There are actually $\frac{n(n+1)}{2}$ different quadratic terms, if we consider that $u_iu_j = u_ju_i$,

but it won't make a difference for what we are going to do). Let U be a vector containing the n^2 possible quadratic terms:

$$U = \begin{bmatrix} u_1u_1 \\ u_1u_2 \\ u_1u_3 \\ \dots \\ \\ u_nu_n \end{bmatrix}$$

We can then represent the problem with a linear system of n^2 variables $\{U_{1,1}, U_{1,2}, ..., U_{n,n}\}$ with the added constraint that $U_{i,j} = u_iu_j$.

We can now express our problem with matrices.

Let $A \in \mathbb{F}_2^{m \times n^2}$ and $b \in \mathbb{F}_2^m$. We want to find $U \in \mathbb{F}_2^{n^2}$ such that $AU = b$.

Example 2 *We can rewrite our previous example as:*

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_{1,1} \\ U_{1,2} \\ U_{1,3} \\ U_{2,1} \\ U_{2,2} \\ U_{2,3} \\ U_{3,1} \\ U_{3,2} \\ U_{3,3} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

5.1 Exercises

Exercise 3 *What is the rate and distance of the Walsh-Hadamard code?*

The rate of a code is defined as $rate = \frac{log(\#\ of\ code\ words)}{\#\ of\ bits\ of\ a\ code\ word}$

The distance of a code is the smallest hamming distance of two code words.

Proof

Here we trivially have that $rate = \frac{log(2^n)}{2^n} = \frac{n}{2^n}$

If we take 2 different code words, they correspond to 2 linear functions $f(x) = x \oplus u$ and $g(x) = x \oplus v$ with $u \neq v$. We have then $f(x) \neq g(x)$ if and only if $(x \odot u) \oplus (x \odot v) = (u \oplus v) \odot x = 1$. Because of the fact that $u \neq v$, $(u \oplus v)$ will never be the zero-vector, and thus, the equation will be satisfied for half of the possible x's which means the distance is $2^n/2$ (and the relative distance is $1/2$). ■

Using the (distance) result of the exercise, we can prove the following

Lemma 8 (Random Subsum Property) *For any $v, u \in \mathbb{F}_2^n$ such that $v \neq u$, we have that $v \odot x \neq u \odot x$ for half of the possible $x \in \mathbb{F}_2^n$.*