Computational Complexity November 12, 2018

Lecture 11 (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Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works.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:

- Zero-knowledge proofs
  - Perfect zero-knowledge interactive proof for Graph Isomorphism.
  - Computational zero-knowledge interactive proof for Graph Coloring.

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 probabilistic 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: $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$. Before proceeding with that, let us state some "easy" statements that we leave for the exercise session:

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

Exercise 2 (Connection to approximation algorithms) *Show that*

$$\textit{GAP 3-SAT is } \mathbf{NP}\textit{-complete} \iff \mathbf{NP} = \mathbf{PCP}(\log n, 1)\,.$$

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.

3 Preliminaries

In order to prove $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$, 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)	0	1	1	0

We can see from the definition that the codewords of the Walsh-Hadamard code are the truth table of all linear functions over $\mathbb{F}_2^n$. Other basic properties of the Walsh-Hadamard code are:

- Its rate is
$$\frac{\log(\#\text{ of code words})}{\#\text{ of bits of a code word}} = \frac{\log(2^n)}{2^n} = \frac{n}{2^n}\,.$$
- Its distance (smallest hamming distance between two codewords) is $2^n/2$ (and the relative distance is $1/2$).

The rate of the code is immediate from its definition. The distance can be seen from the very useful random subsum property:

Lemma 3 (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 $x \in \mathbb{F}_2^n$.*

Proof Select an index j for which $u_j \neq v_j$. We then have

$$\begin{aligned}\Pr[v \odot x \neq u \odot x] &= \Pr\left[(v_j - u_j)x_j = \left(\sum_{i\neq j}(u_i - v_i)x_i \mod 2\right)\right] \\ &= \Pr\left[x_j = \left(\sum_{i\neq j}(u_i - v_i)x_i \mod 2\right)\right] = 1/2\,,\end{aligned}$$

where the last equality follows by first fixing the random outcome of all x_i's with $i \neq j$, which makes $\left(\sum_{i\neq j}(u_i - v_i)x_i \mod 2\right)$ equal some constant b, then $\Pr[x_j = b]$ clearly equals $1/2$. ■

While the Walsh-Hadamard code does not have excellent rate, it has good distance. In addition, we will see in the following subsections that the WH code has two very 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 know 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 $f(x+y) = f(x) + f(y)\ \forall x, y \in \{0,1\}^n \mod 2$. 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 $f(x+y) = f(x) + f(y) \mod 2$.

We see that all linear functions are always accepted.

Definition 4 *Two functions f, g are $(1-\epsilon)$-close if they agree on all, but ϵ-fraction of the inputs.*

Theorem 5 *:* **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.*

The proof of the above theorem is not hard (using the Fourier expansion). However, it requires some additional definitions so we refer the interested reader to the textbook. From the above theorem, we have the following corollary:

Corollary 6 *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* $p \leq \frac{1}{2}$*.*

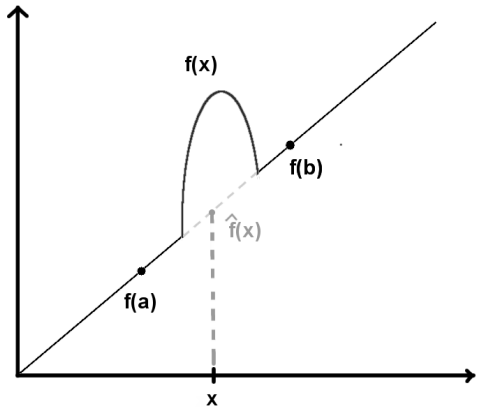
Proof We repeat the test of the theorem $\frac{1}{\delta}$ many times independently.

- 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 7 *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 $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$ 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 $\mathbf{NP} \subset \mathbf{PCP}(poly(n), 1)$, 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 8 *(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:

$$U = \begin{bmatrix} u_1u_1 \\ u_1u_2 \\ u_1u_3 \\ \cdots \\ \\ u_nu_n \end{bmatrix} =: u \otimes u\,.$$

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 $U_{i,j} = u_iu_j$, i.e., that $U = u \otimes u$.

We can now reformulate our problem QUADEQ with matrices: Given $A \in \mathbb{F}_2^{m \times n^2}$ and $b \in \mathbb{F}_2^m$, we want to find $U \in \mathbb{F}_2^{n^2}$ satisfying: (1) $AU = b$ and (2) $U = u \otimes u$ for some $u \in \{0,1\}^n$.

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

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_{1,1} \\ U_{1,2} \\ U_{1,3} \\ U_{2,1} \\ U_{2,2} \\ U_{2,3} \\ U_{3,1} \\ U_{3,2} \\ U_{3,3} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \textit{ and } U = u \otimes u \textit{ for some } u \in \{0,1\}^n.$$

6 Intuition behind PCP verifier

The first verifier that we can think of is a one that gets access to a string $(U, u) \in \mathbb{F}_2^{n^2} \times \mathbb{F}_2^n$ and accepts the string as a valid proof if $AU = b$ and $U = u \otimes u$. Clearly, if a QUADEQ instance is satisfiable by a certain assignment $u_1, \ldots, u_n$, then the verifier accepts the proof (U, u), where $u = (u_1, \ldots, u_n)$ and $U = u \otimes u$. On the other hand, if a QUADEQ instance is not satisfiable, then the verifier rejects any string (U, u). This shows that the suggested verifier is an **NP** verifier for the QUADEQ problem.

Unfortunately, the proof (U, u) is not robust enough in order to use it in a PCP(Poly(n),1) verifier. Remember that a PCP(Poly(n),1) verifier reads Poly(n) random bits based on which (and on the input) it reads a constant number of bits from the proof, and then it finally decides whether or not to accept the proof as a valid one. Therefore, since we are basing our decision on a constant number of bits of the proof, we have to robustify it. One way to do this is by requiring that a large portion of the proof has to depend on a large number of bits of the original proof (U, u), so that a constant number of bits that is randomly chosen from the proof will be likely to be "representative" of the whole original proof (U, u).

Walsh-Hadamard codes is an excellent tool that can help us achieve this requirement. Therefore, instead of getting access to an (n^2+n)-long string (U, u), our PCP verifier will get access to a $(2^{n^2}+2^n)$-long string $(g, f) \in \mathbb{F}_2^{2^{n^2}} \times \mathbb{F}_2^{2^n}$ (g, resp. f, can be thought of as the truth table a function from $\mathbb{F}_2^{n^2}$, resp. from $\mathbb{F}_2^n$, to $\mathbb{F}_2$, we will identify g and f with those functions). A valid proof will consist of a pair of Walsh-Hadamard codes $g = WH(U)$ and $f = WH(u)$, where $(U, u) \in \mathbb{F}_2^{n^2} \times \mathbb{F}_2^n$ satisfies $AU = b$ and $U = u \otimes u$. Thus, the PCP verifier will check (g, f) in 3 steps:

- Step 1: Check that f and g are codewords (i.e., linear functions).
- Step 2: Check that $g = WH(u \otimes u)$, where $f = WH(u)$.
- Step 3: Check that $U = u \otimes u$ satisfies $AU = b$.

Of course, we have to do these steps by making only a constant number of random queries from (f, g), and we have to do it in such a way that the verifier accepts a valid proof with probability 1, and if the problem instance is not satisfiable then the verifier must reject any proof with a probability of at least $\frac{1}{2}$.