Topics in Theoretical Computer Science April 25, 2016

Lecture 9 (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Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works.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:

- Probabilistic Checkable Proofs. **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 $\frac{1}{2}$.
- PCP-Theorem $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.
- Walsh-Hadamard Code: 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 $x \odot u = \sum_{i=1}^{n} x_i u_i \mod 2$.
  - Local testability: there is a test that reads $O(1)$ bits that always accept a code word (linear function) and rejects any function that is not 99.99% close to a linear function with probability at least 99.99%.
  - Local decodability: Given a function $f : \{0,1\}^n \to \{0,1\}$ that is 99.99% close to a linear function $\hat{f}$, we can decode $\hat{f}(x)$, for any $x \in \{0,1\}^n$, correctly with probability 99.98% by reading two bits of f.
- Reformulation of QUADEQ problem: 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) $U = u \oplus u$ for some $u \in \{0,1\}^n$.

Today:

- Proof that $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$ by giving a $\mathbf{PCP}(poly(n), 1)$ verifier for QUADEQ.
- Connection between PCP-Theorem and (hardness of) approximation algorithms.

2 PCP($poly(n), 1$) verifier for QUADEQ

2.1 Intuition

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 $\frac{1}{2}$.

2.2 The Verifier

Given $A \in \mathbb{F}_2^{m \times n^2}$, $b \in \mathbb{F}_2^m$

decide whether there exists a pair (u, U) such that

1. $AU = b$
2. $U = u \otimes u$

The verifier expects as a proof the Walsh-Hadamard encoding of U and u, $WH(U), WH(u)$.

Our verifier will treat the proof as two functions:

$$f : \mathbb{F}_2^{n^2} \to \mathbb{F}_2 \ (WH(U))$$

$$g : \mathbb{F}_2^n \to \mathbb{F}_2 \ (WH(u))$$

Step 1

Do a linearity test that only accepts with probability 0.01 if one of the functions is not 99.99% close to linear.

Step 1 applies this linearity test twice to each of f and g. We have the following:

- If f and g are linear functions, (f, g) passes Step 1 with probability 1.
- If either f or g is not 99%-close to a linear function, (f, g) passes Step 1 with probability of at most $0.01 \leq \frac{1}{2}$, i.e., (f, g) is rejected with a probability of at least $\frac{1}{2}$.
- Step 1 makes $O(1)$ random queries from f and $O(1)$ random queries from g. As the description of f and g contains 2^n and 2^{n^2} bits respectively, we need $O(n^2 + n)$ random bits for Step 1.

Step 2

As f and g are 99%-close to linear functions we assume (between friends) that $g = WH(U)$ and $f = WH(u)$ for some U and u. We need to check that $U = u \otimes u$.

Claim 1 *If $U = u \otimes u$ then $g(r \otimes r') = f(r)f(r')$*

Proof

$$g(r \otimes r') = \sum_{ij} r_i r'_j U_{ij}$$

$$f(r)f(r') = \sum_i u_i r_i \sum_j u_j r'_j = \sum_{ij} r_i r'_j u_i u_j$$

■

Claim 2 *If $U \neq u \otimes u$ then $Pr[g(r \otimes r') = f(r)f(r')] \leq 3/4$*

Proof

$$g(r \otimes r') = \sum_{ij} r_i r'_j U_{ij} = r^t U r'$$

$$f(r)f(r') = \sum_i u_i r_i \sum_j u_j r_j = \sum_{ij} r_i r'_j u_i u_j = r^t (uu^t) r'$$

Now since $U \neq uu^t$, there exist at least one column c_1 in U, and one column c_2 in uu^t such that $c_1 \neq c_2$, and they have the same position in U and uu^t respectively. From the random subsum principle, we know that $r^t c_1 \neq r^t c_2$ for exactly half the possibility of r. Fixing an r such that $r^t U \neq r^t (uu^t)$, we get that for half the possibilities of r', we have that $r^t U r' \neq r^t (uu^t) r'$, hence we get that for at least one fourth of the possible values of (r, r'), $(r \otimes r') \neq f(r)f(r')$. ■

Motivated by the previous two claims, Step 2 of the verifier will repeat the following 3 times:

- Choose $r \in \mathbb{F}_2^n$ and $r' \in \mathbb{F}_2^n$ uniformly and independently.
- Using local decodability, decode $f(r)$, $f(r')$ and $g(r \otimes r')$.
- Check if $f(r) \cdot f(r') = g(r \otimes r')$.

We have the following:

- If $f = WH(u)$ and $g = WH(u \otimes u)$, then according to claim 1, (f, g) passes Step 2 with probability 1.
- If $f = WH(u)$ and $g = WH(U)$ respectively, such that $U \neq u \otimes u$, then:
  - The probability that (f, g) is rejected in this iteration is at least $\frac{1}{4}$ according to claim 2.
  - Thus, the probability that (f, g) is rejected in at least one iteration is at least $1 - \left(1 - \frac{1}{4}\right)^3 > \frac{1}{2}$.
- In each iteration, we need n random bits for the choice of each of r and r'. Moreover, in order to decode $f(r)$, $f(r')$ and $g(r \otimes r')$, we need n, n and n^2 random bits respectively, and we query two bits for each one of them. Therefore, Step 2 uses a total of $3(4n + n^2) = 12n + 3n^2$ random bits, and queries a total of $3(2 + 2 + 2) = 18$ bits from (f, g).

Step 3

Suppose now that $g = WH(U)$ and $f = WH(u)$, where $U = u \otimes u$.

Claim 3 *Let y be a random vector uniformly chosen in $\mathbb{F}_2^m$. If $AU = b$ then $y^t AU = y^t b$ with probability 1. On the other hand, if $AU \neq b$ then $y^t AU \neq y^t b$ with probability $\frac{1}{2}$.*

Proof The claim is trivial for $AU = b$. We simply apply the random subsum property for the case where $AU \neq b$. ■

Notice that $y^t AU = (A^t y)^t U = (A^t y) \odot U = g(A^t y)$ and so $y^t AU$ can be obtained by decoding $g(A^t y)$. Motivated by the previous claim, Step 3 repeats the following twice:

- Choose y uniformly in $\mathbb{F}_2^m$.
- Decode $g(A^t y)$.
- Check if $g(A^t y) = y^t b$.

We have the following:

- If $g = WH(U)$ and $AU = b$, then $g(A^t y) = y^t b$ for all y and (f, g) passes Step 3 with probability 1.
- If $g = WH(U)$ and $AU \neq b$, then:
  - There is a $\frac{1}{2}$ probability that (f, g) will be rejected by this iteration.
  - Therefore, (f, g) will be rejected by Step 3 with a probability of at least $1 - \left(1 - \frac{1}{2}\right)^2 > \frac{1}{2}$.
- Each iteration requires m random bits in order to choose $y \in \mathbb{F}_2^m$, and we need n^2 additional random bits in order to decode $g(A^t y)$. Moreover, two queries from g is needed in order to decode $\hat{g}(A^t y)$. Therefore, Step 3 uses a total of $2(m + n^2) = 2m + 2n^2$ random bits and queries a total of $2(2) = 4$ bits from g.

Completeness of the proposed PCP verifier Suppose that the instance of the QUADEQ problem is satisfiable, there must exist a vector $u \in \mathbb{F}_2^n$ such that $A(u \otimes u) = b$. If we let $f = WH(u)$ and $g = WH(u \otimes u)$, the proof (f, g) will pass all the three steps of the verifier with probability 1.

Soundness of the proposed PCP verifier Suppose that the instance of the QUADEQ problem is not satisfiable, then for any vector $u \in \mathbb{F}_2^n$ we must have $A \cdot (u \otimes u) \neq b$. Let (f, g) be any $(2^n + 2^{n^2})$-long string in $\mathbb{F}_2^{2^n} \times \mathbb{F}_2^{2^{n^2}}$. We have:

- If either f or g is not 99%-close to a linear function, then (f, g) will be rejected by Step 1 with a probability of at least $\frac{1}{2}$.
- If both $f = WH(u)$ and $g = WH(U)$ respectively with $U \neq u \otimes u$, then (f, g) will be rejected by Step 2 with a probability of at least $\frac{1}{2}$.
- If $f = WH(u)$ and $g = WH(U)$ respectively with $U = u \otimes u$, then we must have $AU \neq b$. Therefore, (f, g) will be rejected by Step 3 with a probability of at least $\frac{1}{2}$.
- We conclude that in all cases, there is a probability of at least $\frac{1}{2}$ that (f, g) will be rejected by at least one step of the verifier.
- The total number of random bits that are used is $poly(n, m)$ which is polynomial in the size of the input.
- The total number of queries from the proof (f, g) is $O(1)$.

This shows that QUADEQ $\in \mathbf{PCP}(poly(n, 1))$, which proves that $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n, 1))$.

3 Exercises

Exercise 1 (Taken from David Steurer's course) *We motivated the use of error correcting codes for the proof of the PCP theorem by the claim that query efficient verifiers cannot reliably distinguish between proofs that are close in Hamming distance. In this exercise, you will show this claim.*

Let V be a randomized algorithm (verifier) that given oracle access to a string π (the proof) makes at most q queries to it.

Let $x, \pi \in \{0,1\}^*$ be arbitrary bit strings. Let π' be a bit string obtained by flipping every entry of π with probability ϵ. Show that

$$\Pr[V^{\pi}(x) = 1] \geq \mathbb{E}_{\pi'} \Pr[V^{\pi'}(x) = 1] - q \cdot \varepsilon.$$

Exercise 2 *Given a $\boldsymbol{PCP}(\log(n), 1)$ verifier V for SAT, and an input φ, one can wonder which proof has a maximal probability of being accepted by V along with input φ. Show that this problem is* **NP***-hard to approximate within a factor $1/2$.*

4 Hardness of Approximation

We now explore the fantastic implications the PCP-Theorem has had on our understanding of approximation algorithms. Indeed, the PCP-Theorem shows that Max-3SAT is hard to approximate and then using that as a starting point you can prove your favorite problem to be hard to approximate. The PCP-Theorem has had a similar impact on our understanding of the complexity of approximation as the Cook-Levin Theorem had on our understanding of the limits of exact algorithms.

4.1 Definition of approximation algorithms

Definition 4 *An α-approximation algorithm for a given optimization problem is an algorithm which can be run in polynomial time and which outputs a solution S such that:*

- $\frac{cost(S)}{cost(Optimal\ solution)} \leq \alpha$ *if the problem is a minimization problem ;*
- $\frac{profit(S)}{profit(Optimal\ solution)} \geq \alpha$ *if the problem is a maximization problem.*

It is clear that we will have $\alpha > 1$ for minimization problems and $\alpha < 1$ for maximization problems.

It is possible to prove hardness of approximation results without using the PCP-theorem, as we are going to see in the following example.

4.2 Hardness of traveling salesman problem (TSP)

An instance of TSP is a graph where each edge is labeled with a cost, that must be a positive or a null real number. The expected output is a Hamiltonian path of minimal total cost in this graph.

What we are going to prove is that TSP falls into the worse category of approximation, that is:

Theorem 5 *For any $\alpha \geq 1$, it is NP-hard to get an α-approximation of TSP.*

Proof To prove this, we are going to introduce HAM, which is the problem of determining whether a graph has an hamiltonian cycle. We will use the fact that HAM is NP-hard (we can notice that this directly implies that TSP is NP-hard, since TSP asks for an Hamiltonian cycle in the graph). What we need is a **gap-introducing reduction** from HAM to TSP, that is, a reduction from HAM to TSP. Namely, from a graph G given as input to HAM, it builds a graph G' such that:

- if G has an Hamiltonian cycle, then G' has an Hamiltonian cycle with "low" cost.
- if G has non Hamiltonian cycle, then all Hamiltonian cycles of G' have "high" cost.

The main idea is that knowing a sufficiently good approximation of the solution of TSP for an input that we know to be in either the "high cost" or "low cost" category will enable us to determine in which of the two categories it belongs. This means that a good approximation for TSP can be used to solve HAM because it can distinguish between the two kind of outputs of our reduction.

To do this, we choose a constant M (that we will want to be very high) and we define our reduction as follows:

- The set of nodes of G' is equal to the set of nodes of G.
- G' is complete (all nodes are connected by an edge)
- The weight of an edge e of G' is 1 if e is also an edge of G, and is M otherwise

We denote by n the number of nodes in G and G'. If there exists a Hamiltonian cycle in G, then the same cycle has cost n in G'. Otherwise, any Hamiltonian cycles in G' use at least one edge that does not exist in G, thus it has a cost greater than M.

Now, suppose that we have an $\frac{M}{n}$-approximation of TSP. We denote by S the solution given by our approximation with input G'. If G has a Hamiltonian cycle, then the optimal solution has cost n. As such the cost of S is less than M. Otherwise, the optimal solution has cost greater than M, which means that S also has a cost greater than M. Therefore, we can decide whether G had a Hamiltonian cycle by looking at the cost of S, so our algorithm solves HAM. If M can be arbitrary, we can choose it as large as we want, which proves our theorem. ■

Observation 6 *Although we did not need the PCP-theorem to conclude, we were somehow "lucky" to be able to manipulate the costs of the edges at will. This trick is not a general method since many NP-hard problems, such as looking for the largest clique in a graph are purely combinatorial.*

Observation 7 *We can sometimes get positive results about existence of approximations. For instance, consider the modified version of TSP where the costs along the edges are required to satisfy the triangular inequality, i.e., for $u, v, w \in V$, $cost(u, w) \leq cost(u, v) + cost(v, w)$. The convention is that the cost between two nodes is infinite if there are no edges between them. This problem remains NP-hard. However, a well-known linear 2-approximation consists in performing a depth-first traversal of a minimum spanning tree of the graph (a sub-graph of G, which is a tree and which contains all nodes of G, and for which the sum of the costs of all edges is minimal among all spanning tree of G). A polynomial 1.5-approximation have also been found, and the theoretical limit has not been reached yet.*

4.3 Maximize accept probability

PCP verifiers, themselves, are a source of NP-hard problems. Given a $\mathbf{PCP}(\log(n), 1)$ verifier V for SAT, and an input φ, one can wonder which proof has a maximal probability of being accepted by V along with input φ.

Theorem 8 *It is NP-hard to approximate this problem within a factor of $\frac{1}{2}$.*

Proof The proof of this assertion is very simple, because the gap we need has already been introduced by the definition of a PCP verifier. Let's assume that we have a $\frac{1}{2}$-approximation for this problem. Let Π be the proof constructed by our algorithm with input φ, and p be the probability that Π is accepted by V along with input φ. We can evaluate p in polynomial time by feeding V with all possible random strings since it asks only for $O(\log(n))$ random bits. If $p \geq \frac{1}{2}$, then by the soundness property of V, φ is satisfiable. Otherwise, we have $p < \frac{1}{2}$ which means that the optimal proof cannot have a probability greater than $2p$ of being accepted, and $2p < 1$. By the completeness property of V, φ cannot be satisfiable.

Therefore, running our approximation algorithm on φ, along with an additional polynomial computation tells us whether φ is satisfiable, which means that our verifier solves SAT. ■