E0224 Computational Complexity Theory **Fall 2014**

Lecture 25: November 11, 2014

Lecturer: Chandan Saha *Scribe: Sumant Hegde and Abhijat Sharma*

25.1 Interactive Protocol for TQBF (contd)

In the last lecture we discussed arithmetization and linearization operations on TQBFs and designed a protocol for the same. Then we took the following example

$$\sum_{x_1} L_{x_1} \prod_{x_2} L_{x_1} L_{x_2} \sum_{x_3} \dots L_{x_1} L_{x_2} \dots L_{x_n} P_\varphi = k$$

and traced it for a few iterations, "shaving off" operators from left to right uptill (the first occurrence of) L_{x_2}. We now give an inductive description of the protocol.

Suppose

$$O_{x_1} L_{x_1} O_{x_2} L_{x_1} L_{x_2} O_{x_3} \dots\dots O_{x_n} L_{x_1} L_{x_2} \dots L_{x_n} P_\varphi(x_1, \dots, x_n) = k,$$

where O_{x_i} can be $\sum\limits_{x_i}$ or $\prod\limits_{x_i}$, is the equality that the prover P wants to prove to the verifier V.
The protocol starts with P sending V a suitable prime p so that subsequent computation happens over $\mathbb{F}_p$. At any step (iteration), V asks P to prove an equality of one of the following forms

$$[\sum_{x_j} L_{x_1} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_{j-1}=a_{j-1}} \stackrel{?}{=} B \textbf{ (case 1)}$$

$$[\prod_{x_j} L_{x_1} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_{j-1}=a_{j-1}} \stackrel{?}{=} B \textbf{ (case 2)}$$

$$[L_{x_i} L_{x_{i+1}} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_j=a_j} \stackrel{?}{=} B \textbf{ (case 3)} \tag{25.1}$$

where $a_1, \dots, a_j$ are values in $\mathbb{F}_p$ each randomly chosen in one of the previous rounds.

case 1: For case 1, P sends V the polynomial $r(x_j)$ which is supposed to be

$$[L_{x_1} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_{j-1}=a_{j-1}}.$$

V checks if $r(0) + r(1) = B$ is true and rejects if not. If true, then it picks $a_j \in_{\mathbb{R}} \mathbb{F}_p$, computes $r(a_j)$ and starts next round of iteration with the question:

$$[L_{x_1} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_{j-1}=a_{j-1}} \stackrel{?}{=} r(a_j).$$

case 2: Case 2 is the same as case 1 except that V checks if $r(0) \cdot r(1) = B$ instead of $r(0) + r(1) = B$.

case 3: In Case 3, P sends V a polynomial $R(x_i)$ which is supposed to be

$$[L_{x_{i+1}} \dots L_{x_j} O_{x_{j+1}} \dots\dots P_\varphi]_{x_1=a_1,\dots,x_{i-1}=a_{i-1},x_{i+1}=a_{i+1},\dots,x_j=a_j}.$$

V checks if $a_iR(1) + (1 - a_i)R(0) = B$ is true and rejects if not.
Let us pause here and elaborate on the substep above. Define $Q(x_1, \ldots, x_j) = L_{x_{i+1}} \ldots L_{x_j} O_{x_{j+1}} \cdots P_\varphi$ and so that (25.1) can be rewritten as

$$[L_{x_i} Q(x_1, \ldots, x_j)]_{x_1=a_1,\ldots,x_j=a_j} \stackrel{?}{=} B$$
$$\iff$$
$$[L_{x_i}\{Q(x_1, \ldots, x_j)\}_{x_1=a_1,\ldots,x_{i-1}=a_{i-1},x_{i+1}=a_{i+1},\ldots,x_j=a_j}]_{x_i=a_i} \stackrel{?}{=} B$$
$$\iff$$
$$[L_{x_i}\{Q'(x_i)\}]_{x_i=a_i} \stackrel{?}{=} B$$

where $Q'(x_i) = Q(x_1, \ldots, x_n)_{x_1=a_1,\ldots,x_{i-1}=a_{i-1},x_{i+1}=a_{i+1},\ldots,x_j=a_j}$, which is what $R(x_i)$ is supposed to be. Thus, whenever P is honest,

$$a_iR(1) + (1 - a_i)R(0) = [L_{x_i}\{Q'(x_i)\}]_{x_i=a_i} = B.$$

Continuing, V picks $b_i \in_{\mathbb{R}} \mathbb{F}_p$, computes $R(b_i)$ and starts next round of iteration with the question:

$$[L_{x_{i+1}} \ldots L_{x_j} O_{x_{j+1}} \ldots\ldots P_\varphi]_{x_1=a_1,\ldots,x_{i-1}=a_{i-1},x_i=b_i,x_{i+1}=a_{i+1},\ldots,x_j=a_j} \stackrel{?}{=} R(b_i)$$

□

On a side note, observe that at times $R(x_i)$ can be quadratic in x_i. This can happen specifically when $R(x_i)$ is supposedly a polynomial of the form $[\prod_{x_{i+1}} L_{x_1} \ldots P_\varphi]_{x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$.

The proof of correctness of the protocol follows as in the case of #SAT.
□

25.2 Probabilistically Checkable Proofs (PCPs)

25.2.1 Introduction

There are two ways to view PCPs. Later we will show that the two views are equivalent.

First view: Locally Testable Proof Systems
We know that for a language in NP there is a verifier which takes input x and a certificate ("proof") u, and decides x. The proof has polynomially many bits that can be accessed by the verifier sequentially. Consider now a slightly different scenario where the verifier has random access to the proof, and by just reading very few - say constant number of - bits at random locations in the proof, the verifier can decide x correctly with high probability. Such proof systems are known as *Locally Testable Proof Systems*.
This view may remind us of Hadamard codes (see Lecture 18), since in Hadamard codes it is sufficient to read only two bits of the codeword to retrieve a bit of the message with high guarantee of correctness. Note here that the length of the codeword was exponential in the length of the message.

Second view: Hardness of approximation
This view proves certain inherent limits of approximation algorithms for optimization versions of NP-complete problems. Examples of such problems are maximum independent set, maximum clique size, minimum vertex cover, etc. While we know efficient approximation algorithms for some of them, "there is a limit on the

factors with which the optimal solution can be efficiently approximated"[2], assuming $\mathsf{P} \neq \mathsf{NP}$. Below we discuss some NP-hard optimization problems and analyze their approximation algorithms.

$\mathsf{MAX}-\mathsf{3SAT}$ **(Optimization version of** 3SAT)
Let φ be a 3CNF with m clauses and n variables. For any assignment $u \in \{0,1\}^n$, let $val(\varphi(u))$ be the fraction of clauses satisfied by the assignment u.
Define $val(\varphi) := \max_{u \in \{0,1\}^n} val(\varphi(u))$.
Now $\mathsf{MAX}-\mathsf{3SAT}$ is the task to computing a u from a given φ such that $val(\varphi) = val(\varphi(u))$.
Definition: An algorithm A is said to be a ρ-approximate algorithm for $\mathsf{MAX}-\mathsf{3SAT}$ if for every given φ, A outputs a u such that $\frac{val(\varphi(u))}{val(\varphi)} \geq \rho$.

Example 1 ($\frac{1}{2}$-approximation algorithm for $\mathsf{MAX}-\mathsf{3SAT}$)
Algorithm A:
Input: 3CNF $\varphi(x_1, \ldots, x_n)$.
Step 1: If the all-zeros assignment i.e $u = \{0\}^n$ satisfies at least half of the clauses in φ then output $\{0\}^n$. Otherwise output $\{1\}^n$.□

Proof that A is $\frac{1}{2}$-approximate: Every clause is satisfied by at least one of the assignments $\{0\}^n$ and $\{1\}^n$. Thus either of the assignments satisfies at least half of the clauses in φ. □

Example 2 (2-approximation algorithm for MINIMUM VERTEX COVER)

This being a minimization problem, we define ρ-approximation suitably:
Definition: An algorithm A is said to be a ρ-approximate algorithm for MINIMUM VERTEX COVER if for every given $G(V, E)$, A outputs $S \subseteq V$ such that $\frac{|S|}{|T|} \leq \rho$, where $|T|$ is the size of the minimum vertex cover.
Algorithm B:
Input: $G = (V, E)$.
Step 1: Let $S = \{\}$.
Step 2: If there are no edges in G, go to step 3. Otherwise, pick an arbitrary edge $e(u, v) \in E$ and add both u and v to S. Delete u, v from G along with all the edges incident on u or v. Repeat step 2.
Step 3: Output S.

Proof that B is 2-approximate: The edges we arbitrarily picked one in each iteration form a matching since they are pairwise disjoint. Further, at least one of the endpoints of every edge of the matching is in a minimum vertex cover (by definition) while both the endpoints are added in S by the algorithm. Hence the size of S is at most twice the size of the actual vertex cover, making the algorithm 2-approximate. □
It is conjectured that $\rho = 2$ is the lower bound. The proven lower bound till date is $\rho = 1.36$ due to Dinur and Safra (2005). The proof uses PCP theorem.

Example 3 (7/8-approximation algorithm for $\mathsf{MAX}-\mathsf{3SAT}$)
Algorithm C (randomized):
Input: 3CNF Randomly pick $\varphi(x_1, \ldots, x_n)$.
Step 1: Randomly pick an assignment from $\{0,1\}^n$ and output it.

Proof that C is 7/8-approximate:
Let $X_i = \begin{cases} 1 & \text{if clause } i \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$
It is easy to see that any clause with 3 variables is satisfied by 7 out of all 8 possible assignments. Hence $\mathbb{E}[X_i] = 7/8$ for $1 \leq i \leq m$.

Let X = number of clauses satisfied. Then $X = \sum_{i=1}^{m} X_i$. Further, from linearity of expectations we have

$$\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \sum_{i=1}^{m} (7/8) = 7m/8$$

We can show that probability of satisfying at least a $7/8 - 1/(2m)$ fraction (where m is the number of clauses) is at least $1/poly(m)$. There is also a deterministic algorithm for the problem based on conditional expectation. Also, note that if there were q literals in every clause, we could similarly come up with a $(1 - 1/2^q)$-approximation algorithm.
Theorem: There is a $\frac{7}{8}$-approximation algorithm for $\mathsf{MAX-3SAT}$
Theorem (Håstad): If there is a $(\frac{7}{8} + \varepsilon)$-approximation algorithm for $\mathsf{MAX-3SAT}$ for any $\varepsilon > 0$ then $\mathsf{P} = \mathsf{NP}$.
Proof of the theorem uses PCP theorem, which we will define more formally now.

25.3 Two views of the PCP Theorem (in detail)

The PCP theorem can be formulated in two alternative ways as described briefly above, and we shall state those in more details and prove equivalence of the two different views.

Definition 25.1 ($(r(n), q(n)) - \mathsf{PCP}$ verifier) *Let $L \subseteq \{0,1\}^*$ be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has a $(r(n), q(n)) - \mathsf{PCP}$ verifier if there exists a* probabilistic poly-time algorithm *V, which on input $x (\mid x \mid = n)$, and given access to a proof $u (\mid u \mid \leq 2^{r(n)}.q(n))$,*

1. *uses $r(n)$ random bits and queries the proof u, at $q(n)$ locations, and outputs 1(accept) or 0(reject) accordingly.*
2. *if $x \in L$, $\exists u$ (the correct proof) such that $Pr\{V(x,u) = 1\} = 1$. (Completeness)*
3. *if $x \in L$, then $\forall u$, $Pr\{V(x,u) = 1\} \leq 1/2$. (Soundness)*

Note that the verifier algorithm V has polynomial runnning time, therefore $r(n)$ must be a polynomial in n. Also, since most PCP Theorems can be proved using non-adaptive verfiers, V is generally assumed to be non-adaptive.

Definition 25.2 (Class $\mathsf{PCP}(r(n), q(n))$) *A language $L \in \mathsf{PCP}(r(n), q(n))$ if L has a $(c.r(n), d.q(n)) - \mathsf{PCP}$ verifier, for some constants c and d.*

Having defined the above complexity class of efficient PCP verifiers, we state the first version of the PCP theorem.

25.3.1 PCP Theorem (locally testable proofs version)

Theorem 25.3 $\mathsf{NP} = \mathsf{PCP}(\log n, 1)$

Therefore, this version of the PCP theorem states that every language in NP has a highly efficient PCP verifier.

Remarks:

1. Without loss of generality, the length of the proof u, for a $(r(n), q(n))$-verifier is bounded by $2^{r(n)}.q(n)$, because that is the maximum number of locations that can be queried by the verifier. If it uses $r(n)$ bits, then the index of the location it would query would lie in an interval of size $2^{r(n)}$, and there can be at-most $q(n)$ such intervals that can be covered by the queries.
2. $\mathsf{PCP}(r(n), q(n)) \subseteq \mathsf{NTIME}(2^{r(n)}.q(n))$
   Proof: Suppose L has a $(c.r(n), d.q(n)) - \mathsf{PCP}$ verifier, we need to show that $L \in \mathsf{NTIME}(2^{r(n)}.q(n))$. For any string x, the proof u can be guessed non-deterministically in $2^{O(r(n)}.q(n)$ time and then determinitically verified by running the verifier for all possible choices of $r(n)$ random bits. If the verifier accepts for all $2^{O(r(n)}$ possibilities, then we can accept that $x \in L$, else reject. Thus, this is a $2^{O(r(n)}.q(n)$ non-deterministic algorithm to decide L.[1]
3. The above statement implies one-side of the PCP theorem, that is $\mathsf{PCP}(\log n, 1) \subseteq \mathsf{NP}$. Basically, the more interesting direction to prove is $\mathsf{NP} \subseteq \mathsf{PCP}(\log n, 1)$. We will later prove a weaker version: $\mathsf{NP} \subseteq \mathsf{PCP}(poly(n), 1)$.

25.3.2 PCP Theorem (Hardness of approximation version)

Theorem 25.4 *There exists a constant $\rho(0 < \rho < 1)$, such that for every language $L \in \mathsf{NP}$, there is a deterministic poly-time reduction function f, such that f on input x, outputs a representation of a $3-CNF$ formula ϕ_x, satisfying*

- *if $x \in L, val(\phi_x) = 1$*
- *if $x \notin L, val(\phi_x) < \rho$*

This version of the PCP theorem states the impossibility of the existence of approximation algorithms for the optimization versions of various NP-complete problems, beyond a certain limit, under the assumption that $\mathsf{P} \neq \mathsf{NP}$. Consider the following corollary as an example:

Corollary 25.5 *Let ρ be the constant in the above theorem, then ρ-approximation of $\mathsf{MAX}-\mathsf{3SAT}$ is NP-hard.*

Proof: In other words, we want to prove that any language $L \in \mathsf{NP}$ can be reduced to the ρ-approximation of $\mathsf{MAX}-\mathsf{3SAT}.(0 < \rho < 1)$ Let A be the ρ-approximation algorithm for $\mathsf{MAX}-\mathsf{3SAT}$. Then, given any string $x \in \{0,1\}^*$, we show that we can use the deterministic poly-time reduction described as in **Theorem 25.4** and the algorithm A, to determine whether $x \in L$ or not. The algorithm proceeds as follows:

- Given the string x, apply the reduction f and output ϕ_x such that if $x \in L, val(\phi_x) = 1$ and, if $x \notin L, val(\phi_x) < \rho$.
- Give the boolean formula ϕ_x as input to the algorithm A, which outputs an assignment u which would satisfy at-least a ρ fraction of the maximum satisfiable clauses, $val(\phi_x)$. i.e $val(\phi_x(u))/val(\phi_x) \geq \rho$, where $val(\phi_x(u))$ refers to the fraction of total clauses satisfied by the assignment u.
- Now, if it is the case that $x \in L$, then the assignment u, by definition of A, $val(\phi_x(u)) \geq \rho$.
- otherwise, if $x \notin L$, then as $val(\phi_x) < \rho$, then $val(\phi_x(u))$ can be at-most equal to $val(\phi_x)$, and therefore strictly less than ρ.

Thus, by checking the value of $val(\phi_x(u))$, we can deterministically distinguish between the cases $x \in L$ and $x \notin L$, which proves that the algorithm A is NP-hard. ■

25.4 References

[1] Sanjeev Arora and Boaz Barak, Computational Complexity: A Modern Approach, *Cambridge University Press*, 2009

[2] Hardness of approximation *http://en.wikipedia.org/wiki/Hardness_of_approximation* .