6.893 Approximability of Optimization Problems	November 10, 1999
Lecture 17	
Lecturer: Madhu Sudan	*Scribe: Venkatesan Guruswami*

1 Overview of the Lecture

In this lecture we will first design a PCP verifier for NP languages that tosses polynomially many (in fact $O(n^3)$) random coins, queries the proof in only $O(1)$ places and has completeness 1 and soundness bounded away from 1. More succinctly, this shows that $\mathrm{NP} = \mathrm{PCP}_{1,s}[n^3, O(1)]$ with $s < 1$ (we can of course get $s = 1/2$ at the expense of more, but still $O(1)$, queries to the proof by simple serial repetition of the checking process). The last part of the lecture will define a new NP-complete problem called *Polynomial Constraint Satisfiability*, and assuming an NP-hardness for a "gap" version of this problem, we will design a PCP verifier for NP based on this problem that will use $O(\log n)$ randomness and poly$(\log n)$ queries and achieve completeness 1 and soundness, say, of 1/2. Proof of NP-hardness of this "gap" problem will be the study of the next lecture. With that NP-hardness in place, we will then discuss how to "aggregate" the queries which this verifier makes and then define the notion of "proof composition" and use it to prove the PCP theorem $\mathrm{NP} = \mathrm{PCP}_{1,1/2}[\log n, O(1)]$.

2 $\mathrm{NP} \subseteq \mathrm{PCP}[\mathrm{poly}, O(1)]$

Description of the Verifier: Recall from last lecture that one can represent the NP-complete problem 3SAT as the simultaneous satisfiability of certain polynomials of degree 3. Indeed one can arithmetize each clause C_j of a 3SAT instance ϕ and obtain a degree 3 multivariate polynomial $\hat{C}_j$ over GF(2) such that $C_j(\bar{a})$ is true iff $\hat{C}_j(\bar{a}) = 0$. This is best seen by an example: suppose the clause was $C \equiv (x_1 \vee x_2 \vee \bar{x}_3)$, then $\hat{C}(\bar{x}) = (1 - x_1)(1 - x_2)x_3$.

We would thus like to check that there is an assignment $\bar{a}$ such that $\forall j\ \hat{C}_j(\bar{a}) = 0$.

In order to check that $\hat{C}_j(\bar{a}) = 0$ for all j, the verifier combines these conditions into a single parity condition: it chooses $r_j \in \{0,1\}$ uniformly and independently at random, and simply tests that $\sum_j \hat{C}_j(\bar{a}) = 0$. Note that if $\exists j$ such that $\hat{C}_j(\bar{a}) \neq 0$, then

$$\mathbf{Pr}_{\bar{r}}\big[\sum_j r_j \hat{C}_j(\bar{a}) \neq 0\big] = \frac{1}{2}.$$

Since the $\hat{C}_j$'s are cubic polynomials, by collecting terms we notice that we can write

$$\sum_j r_j \hat{C}_j(\bar{a}) = s_0 + \bar{s}_1 \cdot \bar{a} + \bar{s}_2 \cdot (\bar{a} * \bar{a}) + \bar{s}_3 \cdot (\bar{a} * \bar{a} * \bar{a}) \tag{1}$$

Here $s_0 \in \{0,1\}$ and $\bar{s}_i \in \{0,1\}^{n^i}$ for $i = 1,2,3$, and $\bar{a} * \bar{a} \in \{0,1\}^{n^2}$ is defined by $(\bar{a} * \bar{a})_{ik} = a_i a_k$ for $1 \leq i,k \leq n$ and $\bar{a} * \bar{a} * \bar{a}$ is defined similarly. (All operations in the above Equation are over GF(2).) Note that s_0 and $\bar{s}_i$'s are a function only of the original 3SAT instance and the random string $\bar{r}$, and can be computed by the verifier in polynomial time.

In order to check this by reading very few bits of the proof, the verifier expects the prover to write down the encoding of $\bar{a}$ as per some error-correcting code. In this construction, the proof is expected to be the encoding A of $\bar{a}$ in the Hadamard code, and also the encodings B and C of $\bar{a}$ using the (homogeneous) quadratic and cubic multivariate polynomial codes respectively (note that the Hadamard code is just encoding using all *linear* multivariate, or multilinear, polynomials). Alternatively we can view B, C as the Hadamard encodings of $\bar{a} * \bar{a}$ and $\bar{a} * \bar{a} * \bar{a}$ respectively. Assuming that such tables A, B, C have been written down by the prover, from Equation (1), we only need to check

$$s_0 + A(\bar{s}_1) + B(\bar{s}_2) + C(\bar{s}_3) = 0. \tag{2}$$

This can be accomplished using just three queries to the proof. Of course, the proof might not give the valid tables A, B, C that legally encode the same string $\bar{a}$ as per their respective error-correcting codes. The verifier must thus perform further checks to test that A, B, C are at least close to being valid tables, and then use a self-correction procedure (recall the lecture on self-correcting Hadamard codes) to test the condition of Equation (2). Using the machinery we have already developed for testing and self-correcting Hadamard codes, it will be easy to see that this can also be done using $O(1)$ queries.

We are now ready to formally specify the verifier:

Verifier $V_1^{A,B,C}(\phi, n)$

Input: A 3SAT instance ϕ with n variables and m clauses.

Proof tables: A, B, C where $A : \{0,1\}^n \to \{0,1\}$, $B : \{0,1\}^{n^2} \to \{0,1\}$ and $C : \{0,1\}^{n^3} \to \{0,1\}$.

Accept if and only if all of the following tests succeed:

Linearity Tests:

- Pick $\bar{l}, \bar{l'} \in \{0,1\}^n$ u.a.r and verify $A(\bar{l}) + A(\bar{l'}) = A(\bar{l} + \bar{l'})$.
- Pick $\bar{q}, \bar{q'} \in \{0,1\}^{n^2}$ u.a.r and verify $B(\bar{q}) + B(\bar{q'}) = B(\bar{q} + \bar{q'})$.
- Pick $\bar{c}, \bar{c'} \in \{0,1\}^{n^3}$ u.a.r and verify $C(\bar{c}) + C(\bar{c'}) = C(\bar{c} + \bar{c'})$.

Consistency Tests:

- Pick $\bar{x}, \bar{x'} \in \{0,1\}^n$ u.a.r and verify $\tilde{A}(\bar{x}) \times \tilde{A}(\bar{x'}) = \tilde{B}(\bar{x} * \bar{x'})$
- Pick $\bar{x} \in \{0,1\}^n$ and $\bar{y} \in \{0,1\}^{n^2}$ u.a.r and verify $\tilde{A}(\bar{x}) \times \tilde{B}(\bar{y}) = \tilde{C}(\bar{x} * \bar{y})$

Satisfiability Test:

- Pick $\bar{r} \in \{0,1\}^m$ at random and compute $s_0, \bar{s}_1, \bar{s}_2, \bar{s}_3$ from the $\bar{r}$ and ϕ. Verify that

$$s_0 + \tilde{A}(\bar{s}_1) + \tilde{B}(\bar{s}_2) + \tilde{C}(\bar{s}_3) = 0.$$

Calls to $\tilde{A}, \tilde{B}, \tilde{C}$ above refer to the obvious self-correction procedures: for instance a call $\tilde{A}(\bar{x})$ is computed by picking $\bar{r} \in \{0,1\}^n$ at random and returning $A(\bar{x} + \bar{r}) - A(\bar{r})$, and similarly for $\tilde{B}$ and $\tilde{C}$.

This completes the specification of the verifier. It is clear that if A, B, C are valid encodings of a satisfying assignment $\bar{a}$ then the verifier always accepts.

Note that the verifier clearly makes only $O(1)$ queries to the proof, in fact it makes at most 27 queries, and each time reads one bit from the proof. The amount of randomness used is clearly $O(n^3)$. It therefore only remains to bound the soundness of the verifier.

Lemma 1 *The soundness of the above verifier is at most* 0.99.

By the preceding discussion this immediately gives our main Theorem for today's lecture:

Theorem 2 $\mathrm{NP} \subseteq \mathrm{PCP}_{1,0.99}\big[O(n^3), O(1)\big]$.

Proof of Lemma 1 Suppose some proof A, B, C was accepted by the verifier with probability more than 0.99 and yet the formula ϕ was not satisfiable. We will arrive at a contradiction. By the analysis of the linearity test, if any of A, B, C is not 0.01-close to some linear function, then the Linearity test stage rejects with probability at least 0.01. We may thus assume that there exist linear functions $\hat{A}, \hat{B}$ and $\hat{C}$ that are 0.01-close to A, B, C respectively; say they are the Hadamard encodings of $\bar{a}$, $\bar{b}$ and $\bar{c}$ respectively.

We next argue that if $b_{ij} \neq a_i a_j$ for some i, j, or $c_{ijk} \neq a_i a_j a_k$ for some i, j, k, then the Consistency Tests reject with probability at least 0.19. We prove this using Frievalds' technique that says that if M, N are $n \times n$ $\{0,1\}$ matrices and $M \neq N$, then $\Pr_{x,y \in \{0,1\}^n}[x^T M y \neq x^T N y] \geq 1/4$. Applying this to $M = aa^T$ and $N = (b_{ij})$, we observe that if $b_{ij} \neq a_i a_j$ for some i, j, the first of the consistency tests rejects with probability at least 0.19. This is because that except with probability $2 \times 0.01 = 0.02$, $\tilde{A}(\bar{x}) = \hat{A}(\bar{x})$ and thus except with probability $3 \times 0.02 = 0.06$, the consistency test just checks $x^T M y = x^T N y$; the overall rejection probability when $\bar{b} \neq \bar{a} * \bar{a}$ is therefore at least $1/4 - 0.06 = 0.19$. A similar argument holds for the case when $\bar{c} \neq \bar{a} * \bar{a} * \bar{a}$.

We may therefore assume that A, B, C are 0.01-close to encodings $\hat{A}, \hat{B}, \hat{C}$ of $\bar{a}$, $\bar{b} = \bar{a} * \bar{a}$ and $\bar{c} = \bar{a} * \bar{a} * \bar{a}$ respectively. If $\bar{a}$ is not a satisfying assignment, then with probability 1/2 we have

$$s_0 + \bar{s}_1 \cdot \bar{a} + \bar{s}_2 \cdot \bar{b} + \bar{s}_3 \cdot \bar{c} \neq 0.$$

Since A and $\hat{A}$ are 0.01-close, except with probability 0.02, we have $\tilde{A}(\bar{s}_1) = \hat{A}(\bar{s}_1) = \bar{a} \cdot \bar{s}_1$ and similarly for $\tilde{B}(\bar{s}_2)$ and $\tilde{C}(\bar{s}_3)$. Hence with probability at least $1/2 - 3 \times 0.02 = 0.44$, the "Satisfiability Test" now rejects, giving a contradiction to the fact that the verifier accepted with probability greater than 0.99.

■

Note that the statement of Theorem 2 is non-trivial but does not directly have applications to hardness of approximations because of the exorbitant amount of randomness it uses. Our goal next is to develop a verifier that uses only logarithmic randomness and makes polylogarithmic number of queries. We start out on this goal in the next section.

3 Polynomial Constraint Satisfiability

We first define an optimization problem called *Polynomial Constraint Satisfiability.*

Polynomial Constraint Satisfiability:

Instance: A finite field F, integers m, k, t and $C_1, C_1, \ldots, C_t$ where each $C_j = (A_j; x_1^{(j)}, x_2^{(j)}, \ldots, x_k^{(j)} \in F^m)$ and A_j is an algebraic circuit over F of size s on k inputs.

Objective: Find a polynomial $p : F^m \to F$ of (total) degree d that maximizes the number of j, $1 \leq j \leq t$ such that $A_j(p(x_1^{(j)}), \ldots, p(x_k^{(j)})) = 0$.

The above is a nice and clean algebraic problem. It has a lot of parameters; we will be interested in the following choice of parameters: $d, k, s = \mathrm{poly}(\log t)$; $m = O(\frac{\log t}{\log \log t})$; and $|F| \geq \mathrm{poly}(\log t)$. This gives quite a bit of freedom in picking the parameters, but the choice will always satisfy $|F|^m = t^{O(1)}$. Our interest in this problem stems from the following theorem, which holds for the above choice of parameters, and which will be proved in the next lecture.

Theorem 3 *It is NP-hard to distinguish instances of Polynomial Constraint Satisfiability where all t constraints are satisfiable from those in which at most $t/2$ of the constraints are satisfiable.*

Assuming the statement of the above theorem, we now design a verifier for Polynomial Constraint Satisfiability (which will imply a verifier for all of NP by the above NP-hardness) that uses $O(\log n)$ random coins and makes polylogarithmic number of queries.

Verifier V_2

Input: An instance of Polynomial Constraint Satisfiability.

Proof: A table $\Pi : F^m \to F$ (which is purportedly evaluations of a degree d polynomial p which satisfies all "constraints" A_j).

1. Perform a low-degree test on Π; Reject if the low-degree test rejects. (We assume that if Π passes this test, then except with small probability, say 0.01, there is a unique polynomial $\hat{P}$ of degree d that is $O(1/dk)$-close to Π.)

2. Pick $j \in \{1, 2, \ldots, t\}$ and verify that

$$A_j(\tilde{P}(x_1^{(j)}), \tilde{P}(x_2^{(j)}), \ldots, \tilde{P}(x_k^{(j)})) = 0.$$

   (Here by $\tilde{P}$ we refer to self-correction procedure for low-degree polynomials discussed in an earlier lecture.)

Performance of the Verifier: It is easy to see that if the proof Π is simply the table of evaluations of a degree d polynomial that satisfies all the constraints A_j, then the verifier always accepts, and hence the completeness is 1. For the soundness, suppose every polynomial of degree d satisfies at most $1/2$ fraction of the constraints A_j. Then, in particular the polynomial $\hat{P}$ does not satisfy half the constraints A_j either. Since Π passed the low-degree test, this implies that with probability at least 0.49 the verifier picks a j such that $\hat{P}$ does not satisfy A_j. Since $\hat{P}$ is $O(1/dk)$-close to Π, using a simple union bound and the property of the self-correction procedure, except with probability say 0.01, all k evaluations of $\tilde{P}$ in the second step above yield the correct values of $\hat{P}$, and consequently this test fails if $\hat{P}$ does not satisfy A_j. Thus with overall probability at least 0.48, the verifier rejects. Concluding, the verifier has completeness 1 and soundness 0.52.

We now estimate the randomness and query complexity of the verifier. Both the basic low-degree test and the polynomial self-correction procedure use $O(m \log |F|)$ random bits and query the table Π is $O(d)$ places and each time receive an answer of size $O(\log |F|)$ bits. The basic low-degree test will test for δ-closeness for any constant $\delta > 0$; since we want to test for $O(1/dk)$-closeness, we pick a pairwise independent family of lines and perform the low-degree test with respect to all these lines. This increases the randomness by a factor of 2, while the number of bits queried becomes $O(d^2 \log |F|)$. Since $d, |F| = \mathrm{poly}(\log t)$ and $m = O(\log t / \log \log t)$, the total randomness is still $O(m \log |F|) = O(\log t)$ and the number of bits queried is $\mathrm{poly}(\log t)$.

The verifier thus has completeness 1, soundness bounded away from 1, uses $O(\log t)$ random bits and reads $\mathrm{poly}(\log t)$ bits of the proof. This gives us a second PCP characterization of NP (the first one was given by Theorem 2) as we had promised. The next lecture will go ahead and prove Theorem 3 and thereby complete the proof of the following PCP characterization.

Theorem 4 $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1/2}\big[O(\log n), \mathrm{poly}(\log n)\big]$.