6.841/18.405J: Advanced Complexity Theory April 22, 2001

Lecture 17

Lecturer: Madhu Sudan *Scribe: Claire Monteleoni*

Today's topics:

- Proof of NP $\subseteq$ PCP[$poly\log, poly\log$]
- $\Leftarrow$ Hardness of Polynomial Constraint Satisfaction

1 Gap version of PCS for SAT

The general goal of Polynomial Constraint Satisfaction is to find a function $f : F^m \to F$ satisfying as many constraints of $C_1 \dots C_t$ as possible. Where C_j = algebraic function whose inputs are $f(x_1^{(j)}), \dots, f(x_k^{(j)})$.

The Gap version for SAT is hard. $\Phi \to k, d, F, C_1, \dots, C_t$, such that:

$$\begin{cases} \Phi \in SAT \to \text{all constraints are satisfied by degree } d \text{ polynomial } f. \\ \Phi \notin SAT \to 90\% \text{ of constraints are not satisfied by degree } d \text{ polynomial } f. \end{cases}$$

The idea here is to express satisfiability by finding a function f with the constraint on it's degree (d) that meets the level of satisfaction specified above. The function will take roughly n-bits to specify (cf. a SAT assignment is specified with n-bits).

The goal is to express SAT in terms of algebra, so that we can have some algebraic method of proving satisfiability. The method will be to arithmetize satisfiability. I.e. given Ψ, a candidate formula:

- cf. $\#P$: we counted the number of satisfying assignments for a formula Ψ.
- Now instead, we first fix an assignment, and then count the number of failed clauses of Ψ. If the number of failed clauses is zero, then Ψ is satisfiable.

1.1 Step 1: Arithmetize the assignment

The notion of assignment is: $(w_1, \dots, w_n) \leftarrow (a_1, \dots, a_n) : n = 2^m$, for the variables (w) in Ψ.

- Define the assignment function A, as: $A : \{0,1\}^m \to \{0,1\}$. So $A(i) = a_i$ which is either a 0 or a 1, and thus sets the value of w_i.
- Create a polynomial extension of the assignment function. $\hat{A} : F^m \to F$, such that:

  1. $\hat{A}(x) = A(x)$ if $x \in \{0,1\}^m \subseteq F^m$.
  2. $\hat{A}(x_1, \dots, x_m)$ is a polynomial of degree 1 in each x_i. Note: this does not preclude higher overall degree of the polynomial, due to cross-terms (eg. $\hat{A}(x_1, \dots, x_5) = x_1x_3x_5 + x_4 - x_2x_3$).

  Claim Existence of such an $\hat{A}$.
  For every $A : H^m \to F \quad : H \subseteq F$,
  $\exists \hat{A} : F^m \to F$ such that:

  - $\hat{A}$ extends A. (i.e. $\hat{A}(x) = A(x)$ for $x \in H^m$)
  - degree of $\hat{A}$ in each of its variables is $|H|$ - 1.

Proof [omitted, exercises, algebra textbook] Induction on m (or explicit multivariate interpolation).[1]

Note: writing down $\hat{A}$ is a little more work than writing down A. Prover must write down the table of values that specify the polynomial $\hat{A}$. That is part of the existential step for the prover.

1.2 Step 2: Express the formula as a function

To express the formula Ψ as a function, we define:

- $\Phi : \{0,1\}^m \times \{0,1\}^m \times \{0,1\}^m \times \{0,1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}$
  The input to Φ above is just another notation for a 3CNF clause. So to make this mapping explicit, note that:
  $\Phi(w_{i_1}, w_{i_2}, w_{i_3}, b_1, b_2, b_3) = (A(w_{i_1}) = b_1) \vee (A(w_{i_2}) = b_2) \vee (A(w_{i_3}) = b_3)$
- Create a polynomial extension of Φ. $\hat{\Phi} : F^{3m+3} \to F$, such that $\hat{\Phi}$ extends Φ, and is degree 1 in each of it's variables.

Note: $\hat{\Phi}$ can be computed by the verifier.

1.3 Step 3: Arithmetize satisfiability

Create an arithmetic expression which encodes the satisfiability of Ψ. SAT? : $F^{3m+3} \to F$, such that:
SAT?(Does $\hat{A} : F^m \to F$, satisfy $\hat{\Phi}$)

The function SAT? behaves as follows:
SAT?$(w_{i_1}, w_{i_2}, w_{i_3}, b_1, b_2, b_3) = 0 \implies$

$$\begin{cases} A \text{ satisfies clause } (w_{i_1}, w_{i_2}, w_{i_3}, b_1, b_2, b_3) \text{ OR:} \\ \text{that clause } \notin \Psi \end{cases}$$

The idea here is that we will ask the prover to give us more and more information about the formula we are trying to verify. So now the goal is to create an expression that will be 0 iff the formula is satisfied. What does this expression look like?

$$\text{SAT?} = (\hat{A}(w_{i_1}) - b_1)(\hat{A}(w_{i_2}) - b_2)(\hat{A}(w_{i_3}) - b_3)\hat{\Phi}(w_{i_1}, w_{i_2}, w_{i_3}, b_1, b_2, b_3)$$

Note that by definition of $\hat{A}$ and $\hat{\Phi}$, SAT? will only be at most degree 2 in each variable, so it is a polynomial of degree $2m'$ at most, where $m' = 3m + 3$. Thus its degree is logarithmic in n, where $n = |\Psi|$.

So now we have a well-defined scheme: search for $\hat{A}$ that causes SAT? to be zero somewhere. The question is, on what small subdomain must it be zero? Given $\hat{A}$ and SAT?, how do we know if they are implemented correctly? SAT? is zero on all of $\{0,1\}^{m'}$. There is a good way to check if two polynomials are consistent with one another, but how do we check if a given polynomial is zero on some small subdomain?

Idea cf. the proof of $\#P \subseteq IP$
See Figure 1.

So the idea is that we will slowly make the SAT? polynomial nicer, working on one dimension at a time (i.e. one coordinate at a time), we will make sure that it will be zero for all values on that dimension. So

[1]For the case of m=1, we can use the Polynomial Interpolation Theorem. It is more complicated for higher m.

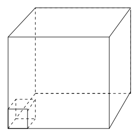

Figure 1: Similar to the #P $\subseteq$ IP proof, in which we were concerned with the smaller cube, $\{0,1\}^n$, where the larger cube was Z_p^n. Now instead we are interested in checking that all values in the smaller cube are zero, which represents $\{0,1\}^{\log n}$, and the larger cube is the field $F^{m'}$.

we define SAT? as:

$$SAT? = \begin{cases} P_0 : F^{m'} \to F \text{ such that } P_0 \text{ is 0 on } \{0,1\}^{m'} \\ \vdots \\ P_i : F^{m'} \to F \text{ such that } P_i \text{ is 0 on } F^i \times \{0,1\}^{m'-i} \\ \vdots \\ P_{m'} : F^{m'} \to F \text{ such that } P_{m'} \text{ is 0 on } F^{m'} \end{cases}$$

Note that these will all be degree $2m'$ polynomials. These will be created by induction on i. In order to compute P_{i+1}, you will need to make oracle calls to P_i. Here is the inductive step: given $P_i(y_1, \ldots, y_i, y_{i+1}, \ldots, y_{m'})$, we need to construct P_{i+1}. We will show how this is done, in the case $m' = 1$. In this case, given $P_0 : F \to F$, we want to implement $P_1 : F \to F$ so that it is:

$$= \begin{cases} 0 \text{ if } (P_0(0) = 0) \wedge (P_0(1) = 0) \\ \text{non-zero otherwise} \end{cases}$$

To implement this, we use the following algebraic gadget: $P_1(y) = yP_0(0) + P_0(1)$

Now note that in the general case, when we move from P_i to P_{i+1}, everything else is fixed, so we can just focus on one variable. Thus we just construct P_{i+1} as follows:

$$P_{i+1}(y_1, \ldots, y_i, y_{i+1}, y_{i+2}, \ldots, y_{m'}) = y_{i+1}P_i(y_1, \ldots, y_i, 0, y_{i+2}, \ldots, y_{m'}) + P_i(y_1, \ldots, y_i, 1, y_{i+2}, \ldots, y_{m'}) \tag{1}$$

Observe that the degree is increasing as we go from i to $i+1$.

1.4 Computing with this construction

First the prover must write down $\hat{A}$. Then in order to specify the function SAT?, the prover must write down P_0, and then each of the higher polynomials P_i.

So how does the verify act to verify these polynomials? The verifier will:

- Check that $\hat{A}$ and SAT? are low degree polynomials, for all $P_0, \dots, P_{m'}$. This amounts to checking that the degree is at least 2 in each variable, i.e. that the total degree of the polynomial is at most $2m'$.
- Check that all the specified relationships hold. This is done as follows. Since the polynomials were each given as a table of values, table look-up can be done to verify that two sides of an equation (1) evaluate to the same value. So pick an i at random and test this. This checking can be done in $O(m')$ =log$n$ time, i.e. one line at a time.[2]

  How much randomness does the verifier need? The verifier performs $O(m')$ tests, and each test needs m'log$|F|$ randomness, so the verifier needs $O((m')^2|F|)$ randomness. But note however that in each case the probability of an error is extremely small. Using the Union Bound, we could instead use the *same* random point to verify all of the equations. The error probability is $\frac{(m')^2}{|F|}$.

1.5 Application back to PCS

Now we must do the final "shrinkwrapping" to show what the above implies about the polynomial constraint satisfaction problem.

First we need to combine all the necessary elements into one single polynomial. Thus in addition to our $P_0, \dots, P_{m'}$, as defined above, we define $P_{-1} = \hat{A} : F^m \to F \underbrace{\leadsto}_{padding} F^{m'} \to F$, where we pad this function with dummy variables, on which it does not really depend, in order to increase the dimension of its domain to m'. Now we define the function $f : F^{m'+1} \to F$ as follows:
$f(i, \hat{y}) = P_i(\hat{y})$ if $i \in \{-1, \dots, m'\}$
Note that f is a polynomial of degree $m' + 1$ in i and at most 2 in each of the variables in the $\hat{y}$ vector.

So from the polynomial constraint satisfaction problem, C_j corresponds to a random choice of a "verifier" that verifies the consistency between P_i and P_{i+1} (equation (1)) for all i.

Question: how large does F have to be? It must be: $|F| \geq (m')^2$. I.e. for $|F| = (\text{log}n)^2$, the size of the proof is $|F|^m = (\text{log}n)^{2\log n} \approx n^{\log\log n}$. So the verifier must write down this much.

1.6 Further enhancements

It is possible to achieve shorter encodings than listed above, by changing base. Above we used the field $F = \{0, 1\}$, and $|F^m| = n \implies m = \text{log}n$. Now instead pick the field $H = \{0, 1, \dots, \text{log}(n-1)\}$. Then $|H^m| = n \implies m = \frac{\log n}{\log\log n}$.[3] For example for $|F| \geq |H|^4 = (\text{log}n)^4 \leadsto F^{m'} \approx n^{12}$, which is better than $n^{\log\log n}$. This will be summarized next lecture.

[2]Schwartz-Zippel

[3]Note that the proof above changes so that instead of the polynomial being degree 2 in each variable, the degree is now $|H|$ in each variable.