CS221: Computational Complexity Prof. Salil Vadhan

Lecture 35: The PCP Theorem and Final Thoughts

12/18 *Scribe: Sasha Schwartz*

Contents

1 Agenda

- Finish PCP proof ideas
- CUE/Poll
- Main points of course

2 Recap

We were in the middle of showing:

Theorem 1 $\mathbf{NP} \subseteq \mathbf{PCP}(\text{polylog}, \text{polylog})$.

Recall that in our PCP construction for 3SAT, the PCP proof is supposed to contain multilinear extension $\hat{A} : \mathbb{F}^{\log n} \to \mathbb{F}$ of a satisfying assignment $A : \{0,1\}^{\log n} \to \{0,1\}$ plus additional information (polynomials in sum-check protocol). Completeness is clear based on previous lectures. From the $\mathbf{IP} = \mathbf{PSPACE}$ analysis we see that soundness will hold if $\hat{A}$ is a low degree polynomial. We sketch in the following section how to verify this, which will complete the proof.

3 Low Degree Testing Problem

Given oracle access to $f : \mathbb{F}^m \to \mathbb{F}$, we would like to test whether f is "close to" a degree d polynomial with few queries and using few random bits. In order to do this we choose a random line in $\mathbb{F}^m$, pick $d+1$ points on this line, and interpolate to see what value f should have on the entire line. We then compare with one other value on the line. This process is more formally described below.

Algorithm

- choose random line ℓ by $x, y \xleftarrow{R} \mathbb{F}^m, \ell(t) = x + yt$
- query $f(\ell(1)), \ldots, f(\ell(d+1))$
- interpolate to find degree d polynomial $q : \mathbb{F} \to \mathbb{F}$ such that $q(i) = f(\ell(i))$ for all $1 \leq i \leq d+1$
- accept if $q(0) = f(\ell(0))$, and reject otherwise

This basic low degree test is fairly complicated to analyze, and we do not have enough time to do so fully. Completeness is clear, because if f is a degree d polynomial then the test accepts with probability one. For soundness we want something like "If f is (often) locally degree d, then f is (close to) globally degree d." This statement is the heart of what's going on in PCP. The following theorem, of which many strengthenings exist, formalizes the desired soundness result.

Theorem 2 (Rubinfield-Sudan) *If this test accepts with probability* $1 - \delta$ *then there exists a degree* d *polynomial* $p : \mathbb{F}^m \to \mathbb{F}$ *disagreeing with* f *is* $O(\delta)$ *fraction of points.*

Returning to the proof of Theorem 1, we run the above test on $\hat{A}$; the total number of queries and randomness is still polylogarithmic. It is enough that $\hat{A}$ is only close to a low degree polynomial instead of equal. This is because, in the PCP, $\hat{A}$ is queried at random points, where it is equivalent to querying the approximating polynomial with high probability.

4 How to get $\mathbf{NP} = \mathbf{PCP}(\log, O(1))$

Here we mention a few of the ideas in proving the actual PCP theorem, where the randomness is logarithmic and the number of bits queried is a constant.

1. Improve above PCP
   - We get $\mathbf{NP} = \mathbf{PCP}(\log, \text{polylog})$ by using low degree extensions instead of multilinear extensions.
   - Parallelize the queries. Instead of making polylog queries of one bit each, make three queries of polylog bits each. This modification is much more substantial than the first one.
   - View PCP as 3-prover MIP. The verifier V takes input x, random bits r, and sends three messages a_1, a_2, and a_3 to its provers with $|a_i| = \text{polylog } n$ for $1 \leq i \leq 3$. Verifier acceptance is then given by a circuit $C_x(r, a_1, a_2, a_3)$ of size polylog n.

2. Proof Composition

   This is used to reduce the query complexity. Instead of reading a_1, a_2, a_3, use another PCP to prove the $C_x(r, a_1, a_2, a_3) = \text{accept}$. There are many technical difficulties in doing this, because normally a PCP verifier requires access to all of the input whereas here the entire point is to avoid reading a_1, a_2, a_3 in their entirety. However, this turns out not to be necessary if input is encoded properly. (For example, in our $\mathbf{PCP}(\text{polylog}, \text{polylog})$ construction, one does not need to read the entire input formula ϕ if instead one is given access to the multilinear extension $\hat{\phi}$.)

   The auxiliary PCP has verifier V_{inner} with input r, uses its own random bits r_{inner}, probes the encoded a_i's for its provers and additionally a PCP that $C_x(r, a_1, a_2, a_3) = \texttt{accept}$. for each set of random bits r. Note that the number of bits read should now be polylogarithmic in the size of the statement "$C_x(r, a_1, a_2, a_3) = \texttt{accept}$", hence $\text{polylog}(\text{polylog}(n)) = \text{poly}(\text{loglog } n)$. The randomness complexity increases by that of the inner verifier, which is only $O(\log(\text{polylog } n)) = O(\text{loglog } n)$.

   We repeat this process over and over to get $\mathbf{NP} \subseteq \mathbf{PCP}(\log, \text{poly} \log\log \cdots \log n)$, where the number of logarithms applied to n is arbitrary.

3. To obtain the final result $\mathbf{NP} \subseteq \mathbf{PCP}(\log, O(1))$ we compose with a different PCP construction which has constant query complexity but $\text{poly}(n)$ randomness complexity. (We can afford such high randomness complexity because now the statements being proven to the inner verifier are extremely short.) The subsequent improvements to the PCP theorem (by Håstad and others) are obtained by constructing inner verifiers with optimized relationships between the soundness and query complexity (while allowing huge randomness complexity, even exponential).

5 Things to take away

1. Computation is not just standard, sequential Turing Machine computation.

   Others include randomized, parallel (hinted at in **NC**), concrete models (circuits etc.), protocols (IP etc.), proofs and their verifications, and quantum computation (which we did not cover).

2. Complexity is not just worst-case time complexity. (computational theory relates resources)

   Other resources include space, circuit size and depth, number of alternations, number of rounds in a protocol, amount of communication, lengths of proofs (proof complexity — guest lecture in reading period), arithmetic operations (algebraic complexity — another guest lecture), and average-case complexity.

3. Problems are not just languages. Other examples are search, functions, counting, and approximation.

What did we actually accomplish?

1. Establishing relations among all of the above. This is how we spent most of the course, and where complexity theory has probably had its biggest successes. The various forms of Savitch's theorem, the many completeness results, and the PCP theorem are all surprising and nontrivial examples.

2. Lower bounds (i.e. negative results). These are generally much harder to come by. We saw two approaches to lower bounds — diagonalization (possibly in combination with some positive simulations, as in $\mathbf{NTIME}(n) \neq \mathbf{TIME}(n)$) and circuit lower bounds (as in PARITY $\notin \mathbf{AC_0}$).

A big theme later in the course was that algebra plus randomization are powerful tools for encoding, simulating, and analyzing computations.