6.841/18.405J: Advanced Complexity Theory April 24, 2002

Lecture 18

Lecturer: Madhu Sudan *Scribe: Nitin Thaper*

Today

- A high level view of NP $\subseteq$ PCP[poly log, O(1)]

  Our main goal will be to look at ways of reducing the number of queries down to a constant.

1 A p-prover 1-round proof system

A p-prover, 1-round proof system consists of a verifier, tossing R private coins and asking 1 question each from p provers. At the end of the protocol, the verifier returns Verdict$(R, a_1, a_2, \ldots, a_p)$ = accept/reject.

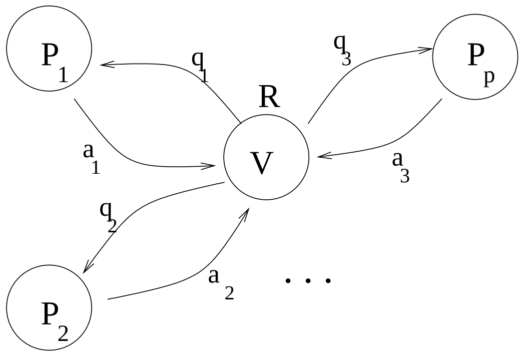

Figure 1: A p-prover, 1-round proof system

We have seen such a model in the context of MIP earlier but today we shall say something stronger about it.

Theorem 1 $\exists\, 3$*-prover proof system for NP where*

- *verifier V tosses* $\log n$ *coins*
- $|a_i| = O(poly \log n)$
- *Completeness = 1, Soundness = 0.1 (arbitrarily small)*

We shall assume this result for the remainder of the lecture (Proving it is part of your problem set)

2 Reducing # queries down further

Observe that asking the question: Is Verdict(R, a_1, a_2, a_3) = accept ?
is equivalent to creating a poly log n sized circuit, $C_R(a_1, a_2, a_3)$ and asking: Does $C_R(a_1, a_2, a_3)$ accept ?

Now in order to reduce the number of queries, intuitively, we would like the provers to commit to their answers, not necessarily reveal them. Then we can run a NP protocol with another set of 3 provers to check if $C_R(a_1, a_2, a_3)$ is true.

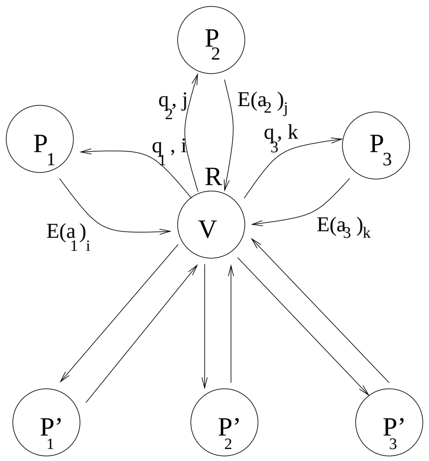

Figure 2: A 6-prover, 1 round protocol for NP. Provers P_1, P_2, P_3 just return a single, encoded bit.

Without giving a formal proof, we claim that this 6-prover protocol is sufficient for any NP problem. Observe that this protocol lets us trade off a constant number of provers with an exponential decrease in the number of query bits. More formally, we've just argued that:

$$\begin{aligned} NP &\subseteq MIP[3(provers), \log n(randombits), \log n(queries)] \\ &\subseteq MIP[6(provers), \log n(randombits), \log\log n(queries)] \end{aligned}$$

This process can be repeated again and again to reduce the number of queries to $\log\log\ldots\log n$. This is still not a constant though! In order to get $O(1)$ queries we have to be clever by trading off random coin tosses against queries.

3 Is NP $\subseteq$ PCP[poly n, O(1)] ?

We shall look at a somewhat orthogonal issue now, namely, is it possible to get a constant number of queries even if we were allowed polynomially many coin tosses. If the answer were yes, we might expect to put together things such that we got our desired result of proving NP $\subseteq$ PCP[log,O(1)].

As usual we'll look at the SAT problem. Consider $\psi \in SAT$ and let π be the proof string.

3.1 Structure of π

Observe that each clause $C_i \in \psi$ can be viewed as a degree 3 polynomial, p_i over F_2, such that

$$p_i = 0 \Leftrightarrow C_i \text{ is satisfied}$$

For example, $C_1 = x_1 \vee \bar{x}_2 \vee x_3$ corresponds to the polynomial, $p_1(x_1, x_2, x_3) = (1 - x_1)x_2(1 - x_3)$. We can derive such a correspondence for all the m clauses.

Given some satisfying assignment a, the honest prover writes down the value of $p(a)$ for every polynomial of degree 3. Since there are $O(n^3)$ possible coefficents, there are $2^{O(n^3)}$ possible polynomials.
Thus, $|\pi| = 2^{O(n^3)}$.

An arbitrary prover, on the other hand, just writes down some function of the polynomials $\{F[p]\}_p$.

The verifier needs to make sure that:

1. $p_j(a) = 0, \forall j = 1..m$
2. $\exists a$ s.t. $F(p) = p(a) \, \forall p$

3.2 Doing Step 1 using constant queries

In order to verify that all clauses are satisfied we need to sonehow combine their corresponding polynomials into a single one, so that a single query reveals the answer with high confidence. A simple sum of all the polynomials wouldn't work since an even number of unsatisfied clauses would still fool the verifier. Instead we do the following:

Pick $r_1, r_2, \ldots, r_m \in_R 0, 1$ and consider the polynomial

$$p_r(x) = \sum_j r_j p_j(x)$$

Observe that this corresponds to taking the sum of a random subset of the polynomials $p_1, p_2, \ldots, p_m$.

Check if:

$$F[p_r] \quad = \quad 0 \tag{1}$$

Repeat this test a few times to get good confidence.

What if the prover is malicious ? In other words, $F(p)$ usually equals $p(a)$ but not on a small subset. Handling this problem is somewhat analogous to self-correction. We'll use the $F(p)$ oracle to get an oracle for $p(a)$.

<u>Given:</u> Oracle F s.t. $\exists a$ s.t. for most p (all but a δ fraction) $F(p) = p(a)$.
Q, a polynomial of degree 3
<u>Goal:</u> Q(a)

Observe that $Q(a) = (P + Q)(a) - P(a)$.

So pick P at random (using $O(n^3)$ random bits) and return

$$Q(a) = F[P + Q] - F[P]$$

It's not hard to see that $Q(a)$ is right with probability at least $1 - 2\delta$.

Thus in equation (1), the verifier should actually be checking $F[P + p_r] - F[P]$ instead of $F[p_{\mathbf{r}}]$.

3.3 Doing Step 2 using constant queries

<u>Given:</u> Oracle F.
<u>Goal:</u> Does $\exists a$ s.t. $Pr_p[F[p] = p(a)] \geq 1 - \delta$

Idea: Use linearity.
Pick P,Q at random. If F[P+Q] = F[P] + F[Q] then

$$\exists a_i \, \exists b_{ij} \, \exists c_{ijk} \; s.t. \, F[P] = \sum p_{ijk} c_{ijk}$$

Note that

$$\#\text{possible F's} = 2^{2^{n^3}}$$

while,

$$\#\text{possible}\, c_{ijk} = 2^{n^3}$$

Phase 2: Use multiplication.
The polynomials can be partitioned into three classes depending on their degree.

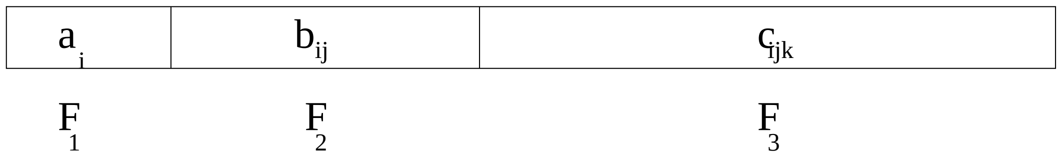

Figure 3: Proof π partitioned into polynomials of different degrees

Pick at random L_1, L_2 of degree 1, Q of degree 2. Test if $F_1[L_1]F_1[L_2] = F_2[Q + L_1L_2] - F_2[Q]$.
Similarly test for degree 3 polynomials by picking $L_1 \in F_1$, $L_2 \in F_2$.

Overall, by reading $\sim 19 = O(1)$ bits from the proof we are still able to get a non-zero probability of detecting cheating. We will not formally prove the corectness of this procedure but a result due to Blum, Ruby and Rubinfeld shows that this is indeed the case.