80240233: Complexity of Computation **Lecture 12**
ITCS, Tsinghua Univesity, Fall 2007 16 November 2007

Instructor: Andrej Bogdanov **Notes by:** Lin Yang

In this lecture we discuss *probabilistically checkable proofs* (PCPs). We first introduce the definition of this paradigm, then give several results that illustrate this paradigm.

Probabilistically checkable proofs are an active area of research, owing to their connections to communication and cryptography, and more surprisingly, to understanding the limitations of approximation alorithms.

1 Definition

To define what PCPs are, we look back to how interactive proofs (IP) work. In an interactive proof, we have a probabilistic polynomial time verifier, and a computationally unbounded prover, and they interact by sending message of polynomial length, so that the verifier is eventually convinced by the prover of the validity of some statement about the input.

A weakness of this system is that the verifier cannot check for inconsistencies among the many potential responses of the prover. We illustrate this point by a specific example. Consider the 3-coloring problem, in which a graph G is given and it is asked whether there exists a valid 3-coloring of the vertices of G. Here is a candidate protocol, which illustrates how the prover can be inconsistent in his answers:

Candidate Protocol for 3-coloring:

V: Choose a random edge from G.
Ask for the colors of the two end points of the edge.

P: Give the colors of the points.

V: If the colors are different, accept.

In this protocol the verifier doesn't know whether the prover always gives the same color for the same vertex. The prover may not be committed to any specific coloring, and he can answer any query by providing two distinct colors. Then the verifier will always accept, even though G may not be 3-colorable.

To eliminate cheating of this kind, we can ask the prover to write out the coloring he should be having in mind into a *proof*. If the graph is not 3-colorable, then there must be at least two endpoints on which the written proof gives the same answer, so the verifier will detect this inconsistency with probability at least $1/m$, where m is the number of edges in the graph.

So the idea is to use written proofs instead of a prover. It might see that we are not making any progress, because this looks just like a usual, non-interactive NP-proof. However, we allow

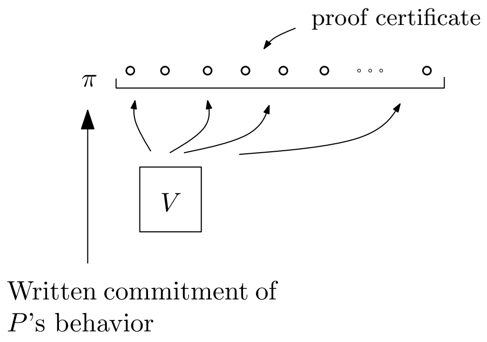

Figure 1: The prover's responses written out

exponential size proof, and allow the verifier to use randomness. This gives a polynomial-time verifier the potential to query as many as $2^{\mathrm{poly}(n)}$ bits in the proof, even though he will actually see only $\mathrm{poly}(n)$ of those bits in any particular run.

In short, PCPs describe interaction between a randomized polynomial time verifier and exponentially long written proof. Denote the verifier and the proof as V and π, respectively. We define the following two parameters for PCPs:

$$q(n) = \text{the number of queries } V \text{ makes to } \pi.$$
$$r(n) = \text{the number of random bits used by } V.$$

Here we give the formal definition:

Definition 1. PCP$(r(n), q(n))$ *is the class of decision problems L such that there exists a randomized oracle polynomial-time algorithm V which, given access to a proof π, on every input x of length n, uses $r(n)$ bits of randomness, makes $q(n)$ queries into π, and*

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \exists \pi : \ \Pr\left[V^{\pi}(x) = 1\right] \geq 2/3 \\ x \notin L \quad &\Rightarrow \quad \forall \pi : \ \Pr\left[V^{\pi}(x) = 1\right] \leq 1/3 \end{aligned}$$

Note that given $r(n)$ randomness and $q(n)$ queries, we can access at most $2^{r(n)} \cdot q(n)$ positions. So the *effective proof length* is $2^{r(n)} \cdot q(n)$, which is in $2^{\mathrm{poly}(n)}$.

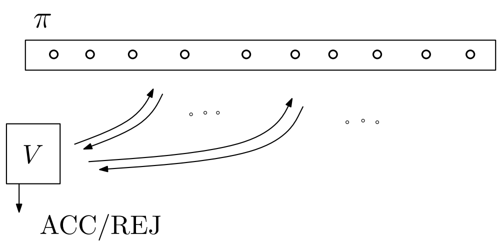

Figure 2: How PCP works

With different configuration of parameters, PCP's power varies greatly. With polynomially many random bits and queries, PCP can simulate interactive proofs, that is

Theorem 2. *IP $\subseteq$ PCP(poly(n), poly(n)).*

To simulate an interactive proof (P, V) by a PCP π, the proof π will contain an entry for each round k sequence of messages $m_1, \ldots, m_k$ sent by the verifier V up to round k. The proof entry $\pi(k, m_1, \ldots, m_k)$ then consists of the response a_{k+1} of the prover P in the next round. The PCP verifier simulates the behavior of V, each time querying the proof π at the location specified by V's questions so far and interpreting the corresponding entry in π as the answer of the prover in the interactive protocol.

Also, as the effective proof length is at most $2^{\mathrm{poly}(n)}$, an NEXP machine can simulate arbitrary PCPs:

Theorem 3. *PCP(poly(n), poly(n)) $\subseteq$ NEXP.*

Surprisingly, this containment is tight:

Theorem 4 (Babai, Fortnow, Lund)**.** *NEXP $\subseteq$ PCP(poly(n), poly(n)).*

Another surprising (and very useful) fact is that NP is equivalent with PCP with $O(\log(n))$ random bits and constant number of queries. This is known as the PCP theorem.

Theorem 5 (Arora, Lund, Motwani, Safra, Sudan, Szegedy)**.** *PCP($O(\log(n)), O(1)$) = NP.*

One implication of the PCP theorem is that if you write the homework in a specific form, Andrej can check whether it is correct with confidence 99% by only looking at a constant number of random places.

We now begin proving Theorem 4.

2 Probabilistically checkable proofs for NEXP

At a very high level, we will try to imitate the proof of $\mathrm{P}^{\#\mathrm{SAT}} \subseteq \mathrm{IP}$. That is, we first construct a boolean predicate to represent the computation of NEXP, then convert the predicate into a polynomial, and at last, use the power of the PCP facilities to check the value of the polynomial.

Recall that, a language L is in NEXP means, there exists non-deterministic Turing Machine N that runs in exponential time, such that N accepts x if and only if $x \in L$. To encode the computation of N into a predicate, we resort to the computation tableau of N on input x, as in the proof of the Cook-Levin Theorem. The computation tableau is a big table whose ith row contains the content of the tape[1] of the Turing Machine at time i, divided into cells. Since in each step the TM is only allowed to change only the portion of the tape near the head of the Turing Machine (the head

[1]We assume N has one tape, which in the last step always contains the answer of the computation in its first cell. Multiple tape NTMs can be simulated by such a NTM while roughly maintaining the running time.

marked on the tape by a special symbol), for each pair of adjacent rows there is a small 2×3 window so that the difference between the rows is limited to this window.

Technically, in any computation the cells of the tableau are occupied by elements of some alphabet Σ whose size is independent of the input x. For convenience we will represent each element of Σ as a binary string and think of the cells of the tableau as occupied by bits rather than elements of Σ. Then the windows that describe changes in the tape grow from size 2×3 to size $2 \times (c/2)$, where c is some constant.

Let n denote the length of x. Since $N(x)$ runs in $2^{\mathrm{poly}(n)}$ steps, it can use at most $2^{\mathrm{poly}(n)}$ cells of the tape, so we can think of the tableau as having dimensions $2^{m/2} \times 2^{m/2}$, where $m = \mathrm{poly}(n)$.

The predicate "N accepts x" is then equivalent to saying that all $2 \times c$ windows in the configuration tableaus are valid transitions with the computation of N on input x. By "valid transitions" we mean that windows in the first row should contain the input x, windows in the middle should either be equal (if the head is not around) or consistent with the operation of N (if the head is around), and the first window in the last row should check that the computation accepted. We can represent all these validity checks by a collection of 2^m boolean predicates corresponding to the windows in the tableau. The idea is illustrated in Figure 3.

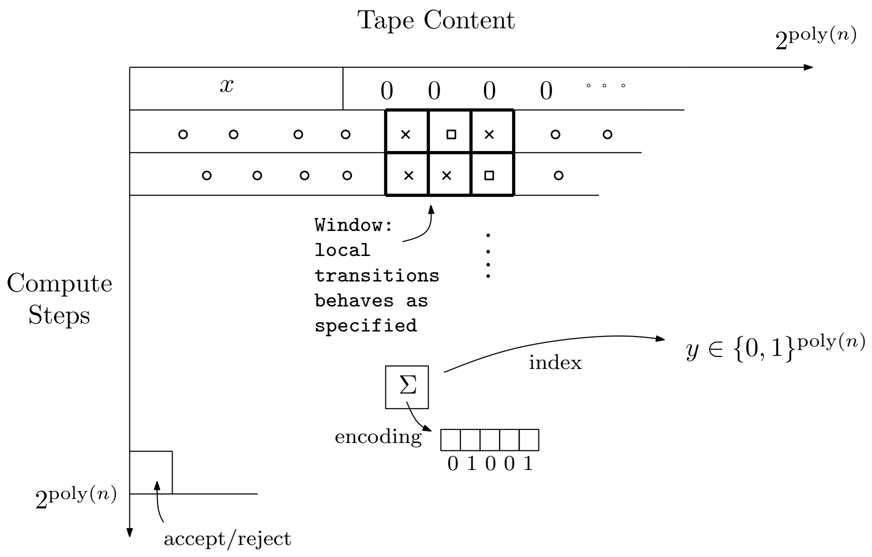

Figure 3: The configuration tableau

Step 1: Encoding As we just argued, there is a collection of 2^m boolean predicates corresponding to windows in the computation tableau so that $N(x)$ accepts iff all these predicates can be simultaneously satisfied. We want to combine all this predicates into a single, efficient predicate VALID. To do this, we introduce a new variable y which will index windows in the computation tableau. Each window can be indexed by its first cell, and there are 2^m cells, so y will take values in $\{0,1\}^m$. Define the predicate VALID as:

$$\text{VALID}(y, b_1, b_2, \dots, b_c) = \begin{cases} 1, & \text{If the transition at the window indexed by } y \text{ is valid} \\ & \text{when the window cells contain the values } b_1, \dots, b_c \\ 0, & \text{otherwise} \end{cases}$$

A boolean formula — in fact, a DNF of length $O(m+c)$ — describing the predicate VALID can be computed efficiently from the description of N and the input x (as well as the bound on the running time of N). We won't worry about the details of this, which are all technical.

A computation of N on input x populates each cell indexed by y of the tableau by a bit; so we can think of it as providing an assignment $A : \{0,1\}^m \to \{0,1\}$. Then N accepts x if and only if this assignment satisfies the VALID predicate for all y:

$$\begin{aligned} x \in L \iff \exists A\ \forall y \in \{0,1\}^m :\ & \text{VALID}(y,\ A(y), \dots, A(y + c/2 - 1), \\ & A(y + 2^{m/2}), \dots, A(y + 2^{m/2} + c/2 - 1)). \end{aligned} \tag{1}$$

To save notation, we will use $\tilde{A}(y)$ to represent the vector of values $(A(y), \dots, A(y+2^{m/2}+c/2-1))$.

Step 2: Arithmetizing Turn the predicate VALID into a polynomial p. We use the same rules as in the proof of $\text{P}^{\#\text{SAT}} \subseteq \text{IP}$: $\bar{u} \to 1-u$, $u \vee v \to u \cdot v$, $u \wedge v \to 1-(1-u)(1-v)$.

Since VALID is a DNF of size $O(m+c)$, the polynomial p will have degree $O(m+c)$. Thus p and VALID have the same value on $\{0,1\}^{m+c}$

$$\forall\ y \in \{0,1\}^m, b_1, b_2, \dots, b_c \in \{0,1\} : \text{VALID}(y, b_1, b_2, \dots, b_c) = p(y, b_1, b_2, \dots, b_c)$$

but p can now be evaluated outside $\{0,1\}^{m+c}$, as in the interactive protocol for $\text{P}^{\#\text{SAT}}$.

The verifier wants to be convinced that for some assignment $A : \{0,1\}^m \to \{0,1\}$, $p(y, \tilde{A}(y)) = 1$ for all $y \in \{0,1\}^m$. Since p is a low-degree polynomial, inspired by the interactive protocol for #SAT, we may ask for a proof of the statement

$$\sum_{y \in \{0,1\}^m} (p(y, \tilde{A}(y)) - 1) = 0$$

by asking for the values of partial sums of the expression $p(y, \tilde{A}(y))$, where the free variables in y are replaced by random elements in a sufficiently large field $\mathbb{F}$.

Step 3: The interactive proof There are several problems with this approach. First, how do we enforce the fact that A should take boolean values over the domain $\{0,1\}^m$? One idea is to modify the expression in a way that asks for the conditions $A(y)(A(y)-1) = 0$ to also be enforced for every y. So the verifier can instead ask for a proof that

$$\sum_{y \in \{0,1\}^m} (p(y, \tilde{A}(y)) - 1) + \sum_{y \in \{0,1\}^m} A(y)(A(y)-1) = 0.$$

Should the verifier be convinced if the prover can prove this statement instead? Not really: Suppose for instance that $p(y, b_1, \ldots, b_c) = 0$ for all inputs. Then the prover can engineer an assignment to the values $A(y) \in \mathbb{F}$ to make the polynomial vanish, e.g., choose $A(0)$ to satisfy the condition $-1 + A(0)(A(0) - 1) = 0$ over $\mathbb{F}$ and $A(y) = 0$ for other ys.

There is a trick we will describe in the next lecture that gets around this problem. The verifier will do the following: First, choose a random $t \sim \mathbb{F}$ (the field should be of size at least 2^m) and then ask the prover to certify that

$$\sum_{y \in \{0,1\}^m} (p(y, \tilde{A}(y)) - 1) \cdot t^{0y} + \sum_{y \in \{0,1\}^m} A(y)(A(y) - 1) \cdot t^{1y} = 0. \tag{2}$$

Here, z denotes the number one would get by reading off the string z in binary, and t^z is the usual powering operation.

Note that if all the summation terms are zero, for every choice of t this expression is zero. Next time we will show that if any one of the terms is nonzero, then for a random choice of t this expression has good chance of being nonzero.

There is another, more important issue at hand. The interactive protocol for #SAT made crucial use of the fact that it was working with a low degree polynomial. The polynomial p is indeed of low degree, but how about $p(y, \tilde{A}(y))$?[2] This expression doesn't even have meaning — A is not even *defined* outside $\{0,1\}^m$! Yet we want to "force" A to behave as a low-degree polynomial, so that the whole expression $p(y, \tilde{A}(y))$ is a polynomial of low degree.

Yet another issue is that at the end of the interactive protocol, the verifier will need to evaluate an expression of the type

$$p(y, \tilde{A}(y)) - 1) \cdot t^{0y} + A(y)(A(y) - 1) \cdot t^{1y}$$

at a random point $y \in \mathbb{F}^m$ to check against the provers claims. But how will the verifier know what A "is" at a random point in $\mathbb{F}$? This is

To answer the last two questions, we have to find a way to specify — and make available to the verifier — the values of the function $A : \mathbb{F}^m \to \mathbb{F}$, where A is a low-degree polynomial which is consistent with the assignment $A : \{0,1\}^m \to \{0,1\}$. The function A will be given to the verifier in the form of an exponentially long table containing all the values $A(y) : y \in \mathbb{F}^m$, as part of the probabilistically checkable proof π. Now it remains to find a way to enforce the fact that the values in this table represent actual evaluations of some low-degree polynomial over $\mathbb{F}$.

3 Multilinear Extension

Recall the problem we are trying to solve: We want the function $A : \mathbb{F}^m \to \mathbb{F}$, which represents the contents computation tableau of $N(x)$, to behave like a low-degree polynomial over $\mathbb{F}$. On the one hand, we want to allow the values of A on $\{0,1\}^m$ to be set arbitrarily in the proof (as we want to

[2]You may think that multiplying by t^z is also an issue, but as we will show next time, this is not a problem – t^z is a low degree polynomial in z.

check the *existence* of tableau that satisfies certain conditions). On the other hand, we want A to be of low degree.

We will use a general procedure that allows us to "extend" any set of values $A(y)$, where $y \in \{0,1\}^m$ into a polynomial $A : \mathbb{F}^m \to \mathbb{F}$ of degree m. This polynomial will be *multilinear*, meaning the individual degree of each variable is one.

A function $A : \mathbb{F}^m \to \mathbb{F}$ is multilinear if there are coefficients $c_S \in \mathbb{F}$ such that

$$q(x_1, x_2, \ldots, x_m) = \sum_{S \subseteq \{1,2,\ldots,m\}} c_S \prod_{i \in S} x_i$$

Claim 6. *Given a set of values $A(y) \in \mathbb{F}$, where $y \in \{0,1\}^m$, it is possible to choose values $A(z) \in \mathbb{F}$ for $z \in \mathbb{F}^m - \{0,1\}^m$ so that A is a multilinear polynomial over $\mathbb{F}$.*

In fact, to make A multilinear, the values $z \in \mathbb{F}^m - \{0,1\}^m$ are uniquely determined, but we won't use this.

Proof. We argue by induction on m. When $m = 1$, $A(y) = yA(1) + (1 - y)A(0)$. For larger m, we extend A via the formula

$$A(y_1, y_2, \ldots, y_m) = y_1 A(1, y_2, y_3, \ldots, y_m) + (1 - y_1) A(0, y_2, y_3, \ldots, y_m)$$

By inductive hypothesis, $A(1, z)$ and $A(0, z)$ can be extended to multilinear polynomials in z, so A can be written as multilinear polynomial in y. $\square$