Avrim Blum **Fall 1993**

Probabilistically Checkable Proofs

Idea: Someone has written out a proof (in some manner) that formula ϕ is satisfiable. (A "proof" is just a string.) You want to check the proof to verify that the formula really is satisfiable. The standard thing is the proof is the satisfying assignment, and you read all the bits. However, you'd like to get away with checking a lot less of the proof – like just a constant number of bits. What you want is that if it's a correct proof, then you accept it; but on the other hand if ϕ is not satisfiable, then no matter what's written, you reject with prob $\geq 3/4$. Sounds impossible.

Note: This is like game in Clique proof, where checked some set of clauses and rejected if found one was not satisfied. Used prob of rejecting to get clique bound.

A $(r(n), q(n))$-*restricted verifier* is a randomized poly time alg that on an instance of size n is allowed to use $O(r(n))$ random bits and query $O(q(n))$ bits of a proof.

A language L is in PCP$[r(n), q(n)]$ if there exists a $(r(n), q(n))$-restricted verifier V such that for all strings x,

- If $x \in L$, then there is a proof Π_x such that $\mathbf{Pr}[V(x, \Pi_x) \text{ accepts}] = 1$.
- If $x \notin L$ then for all Π, $\mathbf{Pr}[V(x, \Pi) \text{ accepts}] \leq 1/4$.

Eg: NP = PCP[0,poly(n)]. RP = PCP$[poly(n), 0]$.

Theorem (ALMSS): NP = PCP$[\log(n), 1]$.

I.e., $O(\log n)$ random bits, and const number of query bits are sufficient to allow you to verify a SAT proof, if it's written correctly.

By the way: easy to see that NP $\supseteq$ PCP$[\log(n), 1]$, since proof only needs to be poly length since for each choice of random bits, there are only a constant number of ways to query constant number of bits.

From before, I said: there exists a poly time transformation T from 3CNF to 3CNF, and a constant ϵ such that: if ϕ is satisfiable then $\phi' = T(\phi)$ is satisfiable. If ϕ is not satisfiable, then at most a $1 - \epsilon$ fraction of the clauses of ϕ' are simultaneously satisfiable.

Claim: The existence of T and the statement about PCP are equivalent.

Transformation implies NP $\subseteq$ PCP$[\log(n), 1]$. WHY? (Give proof of ϕ'. V computes ϕ' and queries $2/\epsilon$ clauses at random. $\mathbf{Pr}[\text{falsely accepts}] \leq (1-\epsilon)^{2/\epsilon} \leq (1/e)^2$.)

Claim: NP=PCP$[\log(n), 1] \Rightarrow$ poly time transformation T.

Idea: use poly time verifier V to get transformation.

Proof: Look at what V does on formula ϕ. Say flips $\leq c_1 lg(n)$ bits and asks $\leq c_2$ questions.

For each possible sequence r of coin flips, make a table showing which sequence of responses causes V to accept, and which causes V to reject. Eg: V accepts on 0000, rejects on 0001, rejects on 0010, accepts on 0011, etc. Now, turn this table into a c_2-CNF formula ϕ_r so that ϕ_r is satisfied *iff* V accepts, as follows.

Formula ϕ_r will say "none of the reject cases occurred." For instance, if a 0001 response corresponds to asking about the 3rd,4th,5th, and 7th bits of the proof, write down $\neg(\bar{x}_3\bar{x}_4\bar{x}_5x_7)$. If 0010 corresponds to asking about bits 3,4,5,8, then AND the clause with $\neg(\bar{x}_3\bar{x}_4x_5\bar{x}_8)$. So, we get up to 2^{c_2} clauses of size c_2.

The formula ϕ' is the AND of all ϕ_r. If ϕ is satisfiable, then there exists a correct proof s.t. V always accepts, so ϕ' is satisfiable.

If ϕ is not satisfiable, then for any proof, at least 3/4 of the strings r cause rejection. This means that for any assignment, on at least 3/4 of the sets of 2^{c_2} clauses, at least one clause is not satisfied (the one that says how V rejected). So, at least a fraction $3/4 \cdot 2^{-c_2}$ of the clauses are not satisfied. ∎

Note: can easily make into 3CNF if desired.

Now: Begin proof that NP $\subseteq$ PCP[$poly(n)$, 1]. I.e., exponential length proof, but only query a constant number of positions, using $poly(n)$ random bits.

This is combined with BFL techniques that acheive NP $\subseteq$ PCP[$polylog(n), polylog(n)$], and recursive shrinking, and other ideas, to get the final result about PCP[$\log(n)$, 1].

To get the result, we need to first solve the following problem. Imagine the prover has written down a string (say, an assignment). We want to ask about the parity of some subsets of the bits of this string. So, imagine for each subset, the prover writes down the parity of that subset. The problem is: how do we know the prover really is writing down information consistent with the parity of bits in some string? I.e., we want to force him to commit to some string, without us checking all the bits. Then we want to make sure we really are getting the right information. To do this, though, we are only allowed a constant number of proof queries.

Another way to think of this: Say the assignment is $(a_1, \ldots, a_n)$. Define the function A on n-bit vectors to be $A(x) = \sum_i a_i x_i (\bmod 2)$. Prover is supposed to write down $A(x)$ for all x, but might write down some other $\hat{A}(x)$.

Get result in two steps:

1. Query so that with good probability $(1 - \epsilon)$, either we have found out the prover is cheating, or at least 90% of values prover has written down are consistent with some linear function A. This is the "Self-testing step."

2. Now, use this to get w.h.p., the right answer for the x's we care about. This is the "Self-correcting step."

Note: This is done in BLR paper.

Do step 1 as follows. Query on random point x. Query on random point y. Then query on $x+y$ (bitwise mod 2). Check consistency. Do several times.

Theorem: Suppose $\hat{G}$ is a $\{0,1\}$ function from Z_2^n to Z_2. Let $\delta < 2/9$. Suppose $\hat{G}$ has property:

$$\mathbf{Pr}_{x,y}[\hat{G}(x) + \hat{G}(y) = \hat{G}(x+y)] \geq 1 - \delta.$$

Then, there is a unique linear G such that G and $\hat{G}$ are $5\delta/2$-close.

We prove this as follows. Define δ' to be such that $\delta'(1-\delta') = \delta$. Note: $\delta \leq \delta' < 3\delta/2$ for $\delta < 2/9$.

Lemma 1: For all a there exists b such that $\mathbf{Pr}_x[\hat{G}(x+a) + \hat{G}(x) = b] \geq 1 - \delta'$. Note: b will be $G(a)$.

Proof: $\mathbf{Pr}_{x,y}[\hat{G}(x+a) + \hat{G}(y) = \hat{G}(x+a+y)] \geq 1 - \delta$.
So, $\mathbf{Pr}_{x,y}[\hat{G}(x+a) + \hat{G}(y) = \hat{G}(x+a+y) = \hat{G}(x) + \hat{G}(a+y)] \geq 1 - 2\delta$.
So, $\mathbf{Pr}_{x,y}[\hat{G}(x+a) + \hat{G}(x) = \hat{G}(y+a) + \hat{G}(y)] \geq 1 - 2\delta$.
Suppose split was: fraction p are b and fraction $1-p$ are $1-b$, where $p \geq 1-p$. So, prob not equal is $2p(1-p) \leq 2\delta$. So, we have $(1-p) \leq \delta'$, and we're done. ∎

Now, we've defined G. Note: This says how can solve problem 1.
Lemma 2: G is linear. Here use $\delta' < 1/3$

Proof: Fix a, a'. So,

$\mathbf{Pr}_x[\hat{G}(x+a)+\hat{G}(x) = G(a)$ and $\hat{G}(x+a')+\hat{G}(x) = G(a')$ and $\hat{G}(x+a)+\hat{G}(x+a') = G(a+a')] \geq 1-3\delta'$.

This implies that $\mathbf{Pr}_x[0 = G(a) + G(a') + G(a+a')] > 0$, which means it's true. ∎

Lemma 3: G and $\hat{G}$ are $5\delta/2$-close.

$$\begin{aligned}
\mathbf{Pr}_z[G(x) = \hat{G}(z)] &\geq \mathbf{Pr}_{z,x}[G(z) = \hat{G}(z+x) + \hat{G}(x) = \hat{G}(z)] \\
&\geq 1 - \delta - \delta' \\
&\geq 1 - 5\delta/2. \quad \blacksquare
\end{aligned}$$

Now, self-correcting (or checking) part is easy. To find $\hat{G}(a)$ you pick a random x and look at $\hat{G}(x+a) + \hat{G}(x)$. Reject proof if ever get different answer. Do a constant number of times to get unlikely you always got the wrong answer (by Lemma 1).

Theorem: NP $\subseteq$ PCP$[poly(n), 1]$.

Proof: It is enough to show that 3SAT $\in$ PCP$[poly(n), 1]$. So, given a 3-CNF formula ϕ, we need to show how a prover can write down an exponential length proof that ϕ is satisfiable which can be checked in just a constant number of bits.

This is what the proof will look like. Say the assignment $A = (a_1, \ldots a_n)$. Define $B = (b_{11}, \ldots, b_{nn})$ where $b_{ij} = a_i a_j$. Define $C = (c_{111}, \ldots, c_{nnn})$ where $c_{ijk} = a_i a_j a_k$. The exponential-length proof will consist of, for each of A, B, and C, and for each subset of the bits in that vector, the parity of the number of ones in that subset.

Another way to think of this is to think of A as defining the linear function on n-bit vectors: $A(v) = \sum_i a_i v_i \pmod 2$. I.e., $A(v) = A \cdot v \pmod 2$. Similarly, $B(v) = B \cdot v$, where v is an n^2-length vector, and similarly $C(v)$. The proof consists of writing down $A(v)$ for all vectors $v \in Z_2^n$, $B(v)$ for all vectors $v \in Z_2^{n^2}$ and $C(v)$ for all vectors $v \in Z_2^{n^3}$.

The verifier now has three tasks:

1. Verify that the prover really has committed to strings A, B, and C, by testing to ensure that 90% of the values claimed to be $A(v)$, for instance, really are consistent with the parities of bits in some fixed string (which we will define to be A). Same for B and C. Then, this can be used in a self-correcting step (to find $A(z)$, compute $A(x) + A(x+z)$ for several random x's).

2. What else has to be checked? (Need to check b_{ij} in the B string committed to really equals $a_i a_j$. Same for c_{ijk}.)

3. Now, use this to check that ϕ is satisfiable.

Last part first: Say there are m clauses. For each clause c_j in ϕ, define the degree-3 polynomial $\tilde{c}_j$ as follows. Say $c_j = x_3 \vee \bar{x}_5 \vee x_6$. Then, $\tilde{c}_j = (1 - x_3)x_5(1 - x_6)$. In other words, for some assignment A, $\tilde{c}_j(A) = 0$ when c_j is satisfied by A, and equals 1 when c_j is not satisfied. This means that a satisfying assignment sets all $\tilde{c}_j$ to 0, and a non-satisfying assignment sets at least one $\tilde{c}_j$ to 1.

Now, notice the following neat fact. For any non-satisfying assignment $A = (a_1, \ldots, a_n)$, if you pick a random bit string $r = (r_1, \ldots, r_m)$, then

$$\mathbf{Pr}_r[\sum_j r_j \tilde{c}_j(A) \pmod 2 \;=\; 1] \;=\; 1/2.$$

(Why?). But, this probability is zero if A is a satisfying assignment. Also, the function $\sum_j r_j \tilde{c}_j(A) \pmod 2$ is just a degree-3 polynomial in the a_i's. Given string r, we can write out this function as:

$$c_0 + \sum_{i \in S_1} a_i + \sum_{(i,j) \in S_2} a_i a_j + \sum_{(i,j,k) \in S_3} a_i a_j a_k \pmod 2$$

for some sets S_1, S_2, S_3 defined by r and ϕ that we can easily figure out.

So, to verify the proof, we pick a random r and ask for the appropriate parities. If the assignment wasn't a satisfying one, we have probability 1/2 of finding that out. We can then repeat a couple of times to drive the failure prob. down further. ■

First part: see prior discussion.

Middle part: Will show how can verify that $b_{ij} = a_i a_j$ for all i, j. Think of B as a matrix. Pick two random column vectors $\mathbf{r}$,$\mathbf{s}$, and compute the bit $\mathbf{s}^t B\mathbf{r}$. Note that this just requires finding out one of the parities. Compare that with the bit $\mathbf{s}^t AA^t\mathbf{r}$. Note that this can be computed with two parities. Now, if B is correct, these will be equal. But: if B is incorrect, these will differ with probability 1/4.