Notes on Prob. Checkable Proof Systems (PCP)

William Gasarch

1 Introduction

Recall the following definition.

Def 1.1 $A \in \mathrm{NP}$ if there exists a polynomial predicate B such that

$$A = \{x : (\exists^p y)[B(x,y)]\}.$$

We want to rewrite this

Def 1.2 $M^{()}$ is an *Oracle Turing Machine- Random access* (henceforth OTM-RA) is an Oracle Turing Machine where the oracle is a string of bits and the requests for the bits is made by writing down the address of the bit. By convention, if the string is s long and a query is made for bit $t > s$ then the answer is 0. We denote a computation with oracle string y by $M^y(x)$. A POTM-RA is an OTM-RA that runs in polynomial time. Note that a POTM-RA can use a string of length 2^{poly} since it can write down that it wants bit position (say) 2^{n^2}. with n^2 bits.

The following is an alternative definition of NP.

Def 1.3 $A \in \mathrm{NP}$ if there is a POTM-RA $M^{()}$ such that
$x \in A \rightarrow (\exists^p y)[M^y(x) = 1]$
$x \notin A \rightarrow (\forall^p y)[M^y(x) \neq 1]$

If $x \in A$ then we think of y as being the EVIDENCE that $x \in A$. This evidence is short (only $p(|x|)$ long) and checkable in poly time. Note that the computation of $M^y(x)$ may certainly use all of the bits of y. What if we restrict the number of bits of the oracle it can look at?

Def 1.4 A *$q(n)$-query POTM-RA $M^{()}$* is a POTM-RA where, on input x of length n, makes at most $q(n)$ bit queries.

Limiting the number of queries would seem to weaken the machines ability. To counter this we will also allow the machine to be randomized.

Def 1.5 Let $q(n)$ and $r(n)$ be monotone increasing functions from N to N and $\epsilon(n)$ be a monotone decreasing function from N to $[0,1]$. $A \in PCP(q(n), r(n), \epsilon(n))$ if there exists a $q(n)$-query POTM-RA $M^{()}$ such that, for all n, for all $x \in \{0,1\}^n$, the following holds.

$x \in A \rightarrow (\exists y)[\Pr_{|z|=r(n)}(M^y(x,r) = 1) = 1]$
$x \notin A \rightarrow (\forall y)[\Pr_{|z|=r(n)}(M^y(x,r) = 1) \le \epsilon(n)]$

Note 1.6

1. The queries are made adaptively. This means that the second question asked might depend on the answer to the first. Hence if $M^y(x,z)$ asks $q(n)$ questions then the total number of questions possible to ask is $2^{q(n)} - 1$. Since there are $2^{r(n)}$ values of z there are a total of roughly $2^{q(n)+r(n)}$ queries that can be asked. Hence we can take $|y| = 2^{q(n)+r(n)}$.

2. The string that you are making bit-queries to can be very long. If the machine is poly time then the string can be exponentially long since to specify a bit takes only poly length. For example, asking for the 1024th bit takes NOT 1024 steps, but only 10 steps (writing down 1024 it base 2).

Example 1.7 SAT is in $PCP(n, 0)$. The y value is the satisfying assignment.

It is known that $SAT \in PCP(O(1), O(\log n))$. This was first shown by [1], though this made much use of [2]. We will not show this proof; however, we will show some weaker results that convey some of the ideas from the proof.

2 An example

We show that $\overline{GI}$ is in $PCP(O(1), n^{0(1)}, 1/4)$. We first show that GI is in AM with private coins and then use the ideas of this protocol to obtain the PCP result. We do not bother defining "AM with private coins" as we are just using it to motivate the PCP for $\overline{GI}$.

$\overline{GI}$ is in Private Coin AM.

1. Input(G, H).
2. Arthur flips two coins. The two sides of the coins are G and H. Let the flips be J_1, J_2. (so $J_1J_2 \in \{GG, GH, HG, HH\}$. Arthur flips $2n \log n$ coins to obtain permutations of the vertices of J_1 and J_2. Permute the graphs as such to obtain J_1', J_2'. Arthur sends (J_1', J_2').
3. Merlin sends back either GG, GH, HG, HH.
4. If Arthur sees that Merlin's response matches J_1J_2 then he accepts. Else he rejects.

If $G \not\equiv H$ then clearly Merlin can tell which graph is which. If $G \equiv H$ then Merlin has only a 1/4 chance of getting it right.

The main idea behind PCP is that Merlin's answers are declared ahead of time to all possible questions.

If $G \not\equiv H$ then the following rather large y will be the string that Arthur queries. The string is a matrix doubly indexed by ALL graphs on n vertices. The entry indexed by (L_1, L_2) has the following:

1. 100 if $L_1 \equiv G$ and $L_2 \equiv G$.
2. 101 if $L_1 \equiv G$ and $L_2 \equiv H$.
3. 110 if $L_1 \equiv H$ and $L_2 \equiv G$.
4. 111 if $L_1 \equiv H$ and $L_2 \equiv H$.
5. 000 if none of the above.

We now describe the PCP protocol.

1. Input(G, H).
2. Arthur flips two coins. The two sides of the coins are G and H. Let the flips be J_1, J_2. (so $J_1J_2 \in \{GG, GH, HG, HH\}$. Arthur flips $2n \log n$ coins to obtain permutations of the vertices of J_1 and J_2. Permute the graphs as such to obtain J_1', J_2'. Arthur makes the query (J_1', J_2') to the string y. (This is really asking about 3 bits.)
3. If the answer is correct then accept. Else reject.

If $G \equiv H$ then clearly the string y described above works.

If $G \not\equiv H$ then any string that you use has probability 1/4 of getting the right answer.

Note that this took 3 bits queries, $2n \log n$ random coin flips, will succeed with prob 1 if $G \not\equiv H$, and will succeed with prob 1/4 if $G \equiv H$.

3 Arithmetician of SAT

Given a 3-SAT formula we will create an arithmetic expression for it. This will differ from other ways you may have seen to do this.

1. We replace variable x with $1 - x$.
2. We replace $\overline{x}$ with x.
3. We replace OR with $\times$.
4. We replace AND with $+$.

For example
$(v_1 \vee \overline{v_2} \vee v_3) \wedge (\overline{v_1} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_0} \vee \overline{v_4} \vee v_6)$
becomes
$(1 - v_1)v_2(1 - v_3) + v_1(1 - v_4)v_5 + v_0v_4(1 - v_6)$.

Why is this a good thing to do? Note that a satisfying assignment will make this expression 0, while a non-satisfying assignment will make this expression nonzero. Hence we can rephrase 3-SAT as the search for roots of an equation.

We need a change of point of view. If there are n variables then we can represent them with $\log n$ bits each. In the above example we have 7 variables so we can think of them as v_{000}, v_{001}, v_{010}, v_{011}, v_{100}, v_{101}, v_{110}. We go further- we think of the assignment as being a function of 3 (more generally $\log n$) variables. So $A(i, j, k)$ will be the value of v_{ijk} (ijk is a number in base 2). We now rephrase the question above:

Is there a function $A : \{0, 1\}^3 \rightarrow \{0, 1\}$ such that

$$\begin{aligned}(1 - A(0,0,1))A(0,1,0)(1 - A(0,1,1)) &+\\ A(0,0,1)(1 - A(1,0,0))A(1,0,1) &+\\ A(0,0,0)A(1,0,0)(1 - A(1,1,0)) &= 0\end{aligned}$$

We would like to write this in another way.
Let χ_i be defined as follows:

1. χ_i will take 5 variables. The first 2 identify what clause it is referring to. The last 3 identify what variable it cares about.

2.
$$\chi_i(c_1, c_2; d_1, d_2, d_3) \begin{cases} 1 & if v_{d_1d_2d_3} \text{ or } \overline{v_{d_1d_2d_3}} \text{ is } i\text{th literal in clause } c_1c_2 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

   We view the clauses as clause 0, clause 1, and clause 2.

3. χ_i can be written as a polynomial in its variables. For example, in our example,

$$\begin{aligned} \chi_3(0,0;0,1,1) = & \ 1 \\ \chi_3(0,1;1,0,1) = & \ 1 \\ \chi_3(1,0;1,1,0) = & \ 1 \end{aligned}$$

   χ_3 is 0 on all other values. Note that

   $\chi_3(c_1, c_2; d_1, d_2, d_3)$ is

   $(1-c_1)(1-c_2)(1-d_1)d_2d_3+(1-c_1)c_2d_1(1-d_2)d_3+c_1(1-c_2)d_1d_2(1-d_3)$.

If there is a satisfying assignment then there is a way to set A such that the following expression is 0 for ALL choices of variables (we will discuss this after the expression).

$\chi_1(0, 1, x_1, x_2, x_3)\chi_2(0, 1, y_1, y_2, y_3)\chi_3(0, 1, z_1, z_2, z_3)\times$
$(1 - A(x_1, x_2, x_3))A(y_1, y_2, y_3)(1 - A(z_1, z_2, z_3))$
$+$
$\chi_2(1, 0, x_1, x_2, x_3)\chi_2(1, 0, y_1, y_2, y_3)\chi_2(1, 0, z_1, z_2, z_3)\times$
$A(x_1, x_2, x_3)(1 - A(y_1, y_2, y_3)A(z_1, z_2, z_3)+$
$+$
$\chi_3(1, 1, x_1, x_2, x_3)\chi_3(1, 1, y_1, y_2, y_3)\chi(1, 1, z_1, z_2, z_3)\times$
$A(x_1, x_2, x_3)A(y_1, y_2, y_3)(1 - A(z_1, z_2, z_3))$

Note that this expression could be written as a polynomial in
$x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3, y_1, y_2, y_3$
$A(x_1, x_2, x_3), A(y_1, y_2, y_3), A(z_1, z_2, z_3)$

We denote this expression $g(x_1, \ldots, z_3, A(x_1, \ldots, z_3))$. (This is not quite correct since A takes 3 variables, not 9.)

Claim 1: If the formula is satisfiable then there is a way to set A such that, for ALL settings of the variables, the expression above is 0.

Set A to correspond to the satisfying assignment. For example, if $v_6 = T$ then set $A(1,1,0) = 1$. For every clause, if the variables are set so that all the χ's are 1, then since A represents a satisfying assignment, the product will be 0. If the variables are set so that some χ is 0, then the expression is 0.

Claim 2: If the formula is satisfiable then there is a way to set A such that, for ANY set of 2^{18} polynomials indexed by $\{0,1\}^{18}$, we have

$$\sum_{x_1=0}^{1} \cdots \sum_{t_3=0}^{1} p_{x_1,\ldots,t_3} g(x_1, \ldots, g_3, A(x_1, \ldots, t_3)) = 0$$

Claim 3: If the formula is not satisfiable then if 2^{18} polys are picked at random it is highly probable that there is no A such that

$$\sum_{x_1=0}^{1} \cdots \sum_{t_3=0}^{1} p_{x_1,\ldots,t_3} g(x_1, \ldots, g_3, A(x_1, \ldots, t_3)) = 0$$

(We do not prove this.)

UPSHOT: What we did for this formula can be done for any formula. Given a formula $\phi(v_1, \ldots, v_n)$ on n variables we can come up with an expression $g(x_1, \ldots, x_{O(\log n)}, A(x_1, \ldots, x_{O(\log n)}))$ that involves an unevaluated A, such that

ϕ is satisfiable iff there is a way to set A such that, for all settings of $x_1, \ldots, x_{O(\log n)}$ g evaluates to 0.

It is likely that you just assumed the arithmetic involved took place over the integers. We will actually do the arithmetic mod p for some p. In that spirit we will also think of A as a map from $F_p^{O(\log n)}$ to F_p; however, we will require that when restricted to $\{0,1\}^{O(\log n)}$ it will output an element of $\{0,1\}$. We will also require A to be of low degree.

For this exposition we will require A to be linear. That is, if you fix all but one variable then its linear. (Actually affine- of the form $ax + b$.) This is NOT how the real proof goes. It is NOT the correct proof. But it will help our exposition and the ideas.

4 A New Problem

Consider the following problem

Given $f(x_1, \ldots, x_L, A(x_1, \ldots, x_L)$ we want evidence that there exists a linear function $A : F_p^L$ to F_p such that strings of 0's and 1's map to $\{0, 1\}$ and

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_L=0}^{1} f(x_1, \ldots, x_L, A(x_1, \ldots, x_L)) = 0$$

The function f is known to the verifier. The values of A are not.

We write this as an PCP protocol. All queries made are assumed to be queries to the oracle. If we ask for a value in F_p then this is actually $\log_2 p$ bit-queries. All arithmetic is done over a finite field to be named later or never.

1. Input $f(x_1, \ldots, x_L, A(x_1, \ldots, x_L))$. Note that this is given symbolically—A is not evaluated.

2. Let
   $\mathrm{LIN}_1(x) = \sum_{x_2=0}^{1} \cdots \sum_{x_L=0}^{1} f(x, x_2 \ldots, x_L, A(x, x_2, \ldots, x_L)$.
   Note that LIN_1 is linear. Ask for $\mathrm{LIN}_1(0)$ and $\mathrm{LIN}_1(1)$. If the sum of the answers is not zero then reject. Otherwise goto the next step.

3. Pick a_1 at random. Ask for the value of $\mathrm{LIN}_1(a_1)$. Since we already know $\mathrm{LIN}_1(0)$ and $\mathrm{LIN}_1(1)$, and LIN_1 is linear, we can, from that, deduce $\mathrm{LIN}_1(a_1)$. Check if the answer the oracle gives is the same answer we can derive for ourselves. If not then reject. If yes then goto next step.

4. Let
   $\mathrm{LIN}_2(x) = \sum_{x_3=0}^{1} \cdots \sum_{x_L=0}^{1} f(a_1, x, x_3, \ldots, x_L, A(a_1, x, x_3, \ldots, x_L))$.
   Note that LIN_2 is linear and we already know $\mathrm{LIN}_2(0) + \mathrm{LIN}_2(1) = \mathrm{LIN}_1(a_1)$. Ask for the values of $\mathrm{LIN}_2(0)$ and $\mathrm{LIN}_2(1)$. If they do not add up to $\mathrm{LIN}_1(a_1)$ then reject. Else goto the next step.

5. Pick a_2 at random. Ask for the value of $\mathrm{LIN}_2(a_2)$. Since we already know $\mathrm{LIN}_2(0)$ and $\mathrm{LIN}_2(1)$, and LIN_2 is linear, we can, from that,

deduce $\mathrm{LIN}_2(a_2)$. Check if the answer the oracle gives is the same answer we can derive for ourselves. If not then reject. If yes then goto next step.

6. Get $a_1, a_2, \ldots, a_{L-1}, a_L, \mathrm{LIN}_1, \mathrm{LIN}_2, \ldots, \mathrm{LIN}_{L-2}, \mathrm{LIN}_{L-1}, \mathrm{LIN}_L$ in a similar fashion. In the last round also ask the oracle for $A(a_1, \ldots, a_L)$ as a further check on the answer.

7. If there was never a reject, then accept.

Each stage the verifier asks $O(1)$ questions and coin flips. There are $O(L)$ rounds. Hence there are $O(L \log_2 p)$ queries and $O(L)$ random coin flips. p will be picked independent of L, so we will have $O(\log n)$ queries and $O(\log n)$ random coin flips.

Why does this work? If there is a linear function $A(x_1, \ldots, x_L)$ such that

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_L=0}^{1} f(x_1, \ldots, x_L, A(x_1, \ldots, x_L)) = 0$$

then the oracle can easily be constructed and answer the questions honestly.

If no such linear function $A(x_1, \ldots, x_L)$ exists then one of the two answers given in the first round is false. Each round keeps testing this more and more and eventually (with high probability) the lie will come out. An important point is that its a fixed oracle, not a player who can plan things.

5 Putting it all together

$PCP(O(\log n), O(\log n), 1/4)$ protocol for SAT.

1. Input $\phi(v_1, \ldots, v_n)$.

2. Using the technique of Section 3 find an expression g in $x_1, \ldots, x_L$ (where $L = O(\log n)$ such that

   ϕ is satisfiable iff there is a linear function $A : F_p^{O(\log n)} \rightarrow F_p$ (where strings of 0's and 1's map to $\{0,1\}$) such that, for all settings of $x_1, \ldots, x_L \in \{0,1\}$, g is 0.

3. Pick 2^L random polynomials indexed by $\{0,1\}^L$.

4. Using the technique of Section 4 to show that $x_1, \ldots, x_L$ $\sum_{x_1=0}^{1} \cdots \sum_{x_L=0}^{1} p_{x_1,\ldots,x_L} g(x_1, \ldots, x_L, A(x_1, \ldots, x_L)) = 0$.

5. If this succeeds output YES, else NO.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45, 1998. Prior version in FOCS92.

[2] S. A. S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45, 1998. Prior version in FOCS92.