CS 6810: Theory of Computing Fall 2021

Lecture 24: Sep 30, 2021

Lecturer: Eshan Chattopadhyay *Scribe: James Meyers*

1 Weak PCP Theorem

We continue our discussion of PCP's and the PCP theorem by proving a weaker result, creating a PCP using polynomial randomness and constant proof access that can decide the NP-complete problem QUAD-SAT.

Definition 1.1. *Given some n, m and a finite (m-length) sequence of n-ary functions $S = \{P_i : \mathbb{F}_2^n \to \mathbb{F}_2\}$, we say $S \in$ QUAD-SAT $\iff \exists a \in \mathbb{F}_2^n (\forall i \leqslant m, P_i(a) = 0)$. In other words, a set of equations over $\mathbb{F}_2^n$ is in QUAD-SAT if it has a solution.*

Theorem 1.2. *QUAD-SAT is NP-complete.*

Proof. A polynomial time verifier for QUAD-SAT simply takes an assignment a, and evalutates each P_i at a, accepting if all values are 0. Given some instance of 3SAT, ϕ, we create a system of quadratic equations. Let n be the number of variables x_n in ϕ and let m be the number of clauses in ϕ. Let variables $a_1 ... a_n$ be members of F_2, creating one for each literal in ϕ.

For each $i \in [1..m]$, there is some clause C_t of the form $x_i \vee x_j \vee \neg x_k$ (negation can appear anywhere or not at all). We turn each $\vee$ into a $+$ and each $\neg x_k$ into $1 - a_k$, producing an equation as follows:

$$c_{t1}a_i + c_{t2}a_j + c_{t3}(1 - a_k) = 1$$
$$1 - c_{t1}a_i - c_{t2}a_j - c_{t3}(1 - a_k) = 0$$

Where in the above, c_{tk} is some variable over $\mathbb{F}_2$ that is only present in this equation (so not repeated in any other translated clauses). Assume there is an assignment making all translated clause equations satisfied. Let the assignment to the 3SAT instance simply take the assignment for the QUAD-SAT instance, turning 1 into true and 0 into false (we ignore the local variables c_{tk}). Then, since each translated clause is of the form $1 - a - b - c = 0$ for some a, b, c, we know at least one of a, b, or c must be equal to 1. For any choice of the constants c_{tk}, then, at least one of the variables a_i can be made to be equal to 1 (or such that $(1 - a_i) = 1$). Then in the corresponding 3SAT clause, the x_i will be true (or $\neg x_i$ will be), and thus ϕ is satisfiable.

Assume ϕ is satisfiable. If we take the satisfying assignment, using 1 for true and 0 for false, we can clearly make each quadratic equation satisfied (with the correct choice for constants c_{tk}).

This reduction is polynomial-time computable, as we create an equation of 6 variables for every 3-ary clause in a 3SAT instance. Thus, QUAD-SAT is NP-complete. □

Theorem 1.3. $\exists c \in \mathbb{N}(NP \subseteq PCP(n^c, O(1)))$

Proof. Given some $L \in$ NP. Because QUAD-SAT is NP-complete, we know $L \leqslant_P$ QUAD-SAT. So fix some polynomial computable reduction r such that $\forall x(x \in L \iff f(x) \in$ QUAD-SAT$)$. Now, we create a PCP system using polynomial randomness and constant queries to decide the QUAD-SAT instance $r(x)$ with perfect completeness and $\frac{1}{2}$ soundness.

Intuitively, we could think to take as proof a simple assignment vector $\mathbf{a}$ that we could then use to check whether each QUAD-SAT equation evaluates to 0. However, we could never read an entire assignment with a constant number of bits, so we must get a little more creative.

Given some collection of quadratic equations $P_1, P_2, ..., P_m$ for some m, each acting on n variables. Quadratic equations over $\mathbb{F}_2$ can be written in a general form as follows: $\Sigma_{i,j \leqslant n} c_{i,j} a_i a_j$. So any quadratic equation over n variables can then be written as a string of n^2 bits e—we can let for any k, $e[k]$ be number representing the coefficient $c_{k//n, k\%n}$ which is multiplied by $a_{k//n} a_{k\%n}$, with $//$ representing integer division. So, for any n, there are $2^{O(n^2)}$ quadratic equations on n variables operating over $\mathbb{F}_2$.

In our PCP system, we treat proofs π as the enumeration of each of the $2^{O(n^2)}$ quadratic equations evaluated on some assignment $\mathbf{a}$, concatenated with the enumeration of each of the 2^n linear functions evaluated on $\mathbf{a}$. Then in general, to test whether an assignment satisfies our equations P_1 through P_m, we can simply check if their evaluation at $\mathbf{a}$ is zero for each. However, we must come up with creative approaches to 1) ensure that the proof π does indeed encode the evaluation of quadratic/linear equations on a singular assignment to n variables, and 2) that the system of equations is actually satisfied. Both of these must also be checked with polynomial use of randomness, and a constant number of queries.

So, the first bit is the value of the quadratic function with the lexicographically-smallest bit representation, evaluated with variables $a_1, a_2, ..., a_n$ assigned based on the values of the n-element vector $\mathbf{a}$. In general, this is a pair of Walsh-Hadamard encodings of $\mathbf{a} \otimes \mathbf{a}$ and $\mathbf{a}$. With the strings $WH(\mathbf{a} \otimes \mathbf{a}), WH(\mathbf{a})$ representing these encodings, we can think of them as truth tables to functions $f : \{0,1\}^{n^2} \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$. In general for such an encoding e, the value $e(\mathbf{x}) = \mathbf{u} \odot \mathbf{x}$ for any $\mathbf{x}$ and some $\mathbf{u}$ in $\{0,1\}^{|x|}$. Therefore, clearly f and g should be linear functions (so for any $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, $f(\mathbf{x}) + f(\mathbf{y}) = f(\mathbf{x} + \mathbf{y})$). This property is beneficial in that it allows us to check that the prover did indeed send a Walsh-Hadamard encoding of some assignment: if either f or g is not linear, π cannot be such an encoding. And if they are both linear, then each represents a Walsh-Hadamard encoding of evaluations of quadratic/linear functions, each on **some** assignment (knowing that they are both linear does **not** ensure that they encode evaluations of the **same** assignment—in fact, f being linear doesn't ensure that it encodes evaluations of quadratic functions. A linear f could simply represent evaluations of all linear functions on an assignment to n^2 distinct variables!).

To test the linearity of f and g with only constantly many queries, we can apply the BLR test. In other words, we randomly choose two n-ary functions over $\mathbb{F}_2$, say q_i and q_j. If we have that $f(\mathbf{q}_i) + f(\mathbf{q}_j) \neq f(\mathbf{q}_i + \mathbf{q}_j)$, then we know that f cannot be linear. This test can be done in constant queries, because for any function representation $\mathbf{q}$, we know that its evaluation on the assignment $\mathbf{a}$ should be located at the $\mathbf{q}^{\text{th}}$ bit of π.

The probability that we reject a non-linear function f depends on its "distance" from any linear function. We can utilize the notion of the Hamming distance between truth tables of functions to define ρ-close functions:

Definition 1.4. *For any functions $f : \{0,1\}^n \to \{0,1\}, g : \{0,1\}^n \to \{0,1\}$, we say f and g are ρ-close if*

$$Pr_{\boldsymbol{x} \in \{0,1\}^n}[f(\boldsymbol{x}) = g(\boldsymbol{x})] \geqslant \rho$$

The following claim is proved in the lecture 21 notes:

Claim 1.5. *For any $f : \{0,1\}^n \to \{0,1\}$, there is some linear function that is ρ-close to f if*

$$Pr_{\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n}[f(\boldsymbol{x} + \boldsymbol{y}) = f(\boldsymbol{x}) + f(\boldsymbol{y})] \geqslant \rho$$

Claim 1.6. *We can create a test that makes a constant number of queries, say for $\delta = .01$ that ensures any f that is $(1-\delta)$-close to a linear function is rejected with probability at least $\frac{1}{2}$.*

Proof. Given some f, assume it is at most $(1-\delta)$-close to a linear function for some $\delta \in (0, \frac{1}{2})$. Then the probability, ϵ, that a single BLR test accepts f is the probability that a randomly sampled $\mathbf{x}$ and $\mathbf{y}$ satisfy the linearity condition, which is smaller than $(1-\delta)$. For any $\delta \in (0, \frac{1}{2})$, we have that because $\epsilon \leqslant (1-\delta)$ and $\delta \in (0, \frac{1}{2})$, $\epsilon \geqslant \delta$. Then because in general, $1 + x \leqslant e^x$, we have $(1+x)^k \leqslant e^{xk}$ for positive k and $(1+x)$. So, we know

$$\begin{aligned}
(1-\epsilon)^{\frac{2}{\delta}} &\leqslant e^{\frac{-2\epsilon}{\delta}} \\
&\leqslant e^{-2} && \frac{\epsilon}{\delta} \geqslant 1 \text{ and } e^x \text{ monotone decreasing for } -x \\
&< \frac{1}{2}
\end{aligned}$$

Thus, we can design a BLR test rejecting any function f that is not $(1-\delta)$-close to a linear function with probability at least $\frac{1}{2}$. We repeat the random sampling of $\mathbf{x}$ and $\mathbf{y}$ for $\frac{2}{\delta}$ iterations. So then we can create a test that makes a constant number of queries, say for $\delta = .01$ satisfying these properties. □

So, the PCP verifier V should first test the linearity of f and g (as defined by the proof π). Then, we have check that the evaluations of each of the $P_1, P_2, ..., P_m$ on the assignment $\mathbf{a}$ are all 0. So, in general, we hope to be able to find the value $f(\mathbf{x})$ for any $\mathbf{x}$. Though we can reject any functions greater than .99-close to linearity with high probability, it could be the case that f has errors on some of the P_i functions. Thus, we need to ensure that we can ascertain the value of $f(\mathbf{x})$ with high probability and a constant number of queries.

The following claim is easy to obtain based on the distance of the Welsh-Hadamard code.

Claim 1.7. *For any $\bar{f}, \bar{g}, f$ with f .99-close to $\bar{f}$ and $\bar{g}$ and both $\bar{f}$ and $\bar{g}$ linear, it must be the case that $\bar{f} = \bar{g}$.*

We use the above claim to ensure that we can retrieve the unique value of $\bar{f}(\mathbf{x})$ with high probability. For any $\mathbf{x} \in \{0,1\}^n$, we do the following. Choose some randomly sampled $\bar{x} \in \{0,1\}^n$. Then by linearity of $\bar{f}$, it should be the case that $\bar{f}(x) = \bar{f}(\bar{x} + x) + \bar{f}(\bar{x})$ (every $x \in \mathbb{F}_2$ is its own additive identity). By the .99 closeness of $\bar{f}$ and f, we know that the probability that $\bar{f}(x) \neq f(x) \leqslant .01$. Thus by union bound, the probability that $f(x) \neq f(\bar{x} + x) + f(\bar{x})$ is at most .02. So, from now on, when we refer to $f(\mathbf{x})$ or $g(\mathbf{x})$, we can assume that we are utilizing this self-correction and have the correct value with good probability.

After testing the linearity of f and g and ensuring that we can decode them with high probability, we refer to f and g instead of the $\bar{f}$ and $\bar{g}$ (i.e., from now on, we assume f, g are both linear).

Note that ensuring that f is linear function only ensures that f on input $q = (q_{i,j})$ evaluates to $\Sigma_{i,j} b_{ij} q_{ij}$ for some matrix B (b_{ij} being the (i,j)'th entry). What we really want is to ensure that f resembles a function of the form $\Sigma_{i,j} a_i a_j q_{ij}$, where a_i and a_j are defined by the assignment $\mathbf{a}$. In other words, right now we can be confident that $f(\mathbf{x})$ for any $\mathbf{x}$ is equal to $\mathbf{u}B\mathbf{v}^T$, for some matrix B, where $\mathbf{u}$ and $\mathbf{v}$ are used to generate coefficients q_{ij}. We want to know whether $f(\mathbf{x})$ is close to $\mathbf{u}\mathbf{a}^T\mathbf{a}\mathbf{v}^T$. Both $\mathbf{u}\mathbf{a}^T$ and $\mathbf{a}\mathbf{v}^T$ can be encoded as linear transformations on $\mathbf{a}$, so we consult our second Walsh Hadamard encoding.

For linear functions l_u and l_v, $l_u(\mathbf{x})l_v(\mathbf{x}) = (l_u \cdot l_v)(\mathbf{x})$. With f encoding quadratic equations evaluated on the assignment $\mathbf{a}$ and g encoding linear equations evaluated on that assignment, it should be the case that $f(\mathbf{x} \otimes \mathbf{y}) = g(\mathbf{x})g(\mathbf{y})$ for any $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$. The tensor product $\mathbf{x} \otimes \mathbf{y}$ represents coefficients of a quadratic equation on n variables (and thus the multiplication of two linear functions represented by $\mathbf{x}$ and $\mathbf{y}$). Since we know that f and g are both linear, we know that $\forall \mathbf{x} \in \{0,1\}^{n^2}$, $f(\mathbf{x}) = \mathbf{W} \odot \mathbf{x}$ for some $\mathbf{W}$, and $\forall \mathbf{x} \in \{0,1\}^n$, that $g(\mathbf{x}) = \mathbf{a} \odot \mathbf{x}$ for some $\mathbf{u}$. So, we must check that $\mathbf{W} = \mathbf{a} \otimes \mathbf{a}$.

We can check this via 3 iterations of the following procedure:

Choose $\mathbf{x}$ and $\mathbf{y}$ randomly from $\{0,1\}^n$. Query the proof π to retrieve the values $f(\mathbf{x} \otimes \mathbf{y})$, $g(\mathbf{x})$ and $g(\mathbf{y})$. Finally, ensure that $f(\mathbf{x} \otimes \mathbf{y}) = g(\mathbf{x})g(\mathbf{y})$, rejecting otherwise. For any linear g and f, we have $g(\mathbf{x})g(\mathbf{y}) = (\mathbf{a} \odot \mathbf{x}) * (\mathbf{a} \odot \mathbf{y}) = (a_1x_1 + ... + a_nx_n) * (a_1y_1 + ... + a_ny_n) = (a_1^2x_1y_1 + a_1x_1a_2y_2 + ... + a_n^2x_ny_n) = (\mathbf{a} \otimes \mathbf{a}) \odot (\mathbf{x} \otimes \mathbf{y}) = f(\mathbf{x} \otimes \mathbf{y})$. So, for any linear f and g with $\mathbf{W} = \mathbf{a} \otimes \mathbf{a}$, we reject with probability 0.

Assume $\mathbf{W} \neq \mathbf{a} \otimes \mathbf{a}$. We have that

$$\begin{aligned} f(l_x \cdot l_y) - g(l_x) \cdot g(l_y) &= \Sigma_{i,j} W_{i,j} x_i y_j - \Sigma_{i,j} a_i a_j x_i y_j \\ &= \Sigma_{i,j} (W_{i,j} - a_i a_j) x_i y_j \end{aligned}$$

With fixed values for $\mathbf{x}$ and $\mathbf{y}$, we know that there must be some i, j such that $B_{i,j} - a_i a_j \neq 0$. As we showed earlier, for any linear f and g that are not equal, they differ on at least half of their inputs. Then because $\mathbf{W} \neq \mathbf{a} \otimes \mathbf{a}$ (and both encode linear transformations), with probability $\frac{1}{2}$ a random $\mathbf{x}$ satisfies $\mathbf{x}\mathbf{W} \neq \mathbf{x}(\mathbf{a} \otimes \mathbf{a})$. Then, with a further $\frac{1}{2}$ probability (so total $\frac{1}{4}$), a randomly sampled $\mathbf{y}$ satisfies $\mathbf{x}\mathbf{W}\mathbf{y} \neq \mathbf{x}(\mathbf{a} \otimes \mathbf{a})\mathbf{y}$. So for 3 independent trials, if $\mathbf{W} \neq \mathbf{a} \otimes \mathbf{a}$, we reject with probability $\frac{3}{4}$.

Finally, if we have not rejected yet, we know that f and g are both linear and do indeed encode the evaluation of all quadratic and linear equations over $\mathbb{F}_2$ on some assignment $\mathbf{a}$. So, now we need only check that the system of equations is satisfiable. Ideally, we could test every of the m quadratic equations at $\mathbf{a}$, but this would require m queries, which need not be constant. Instead, we can test a single random linear combination of all of the quadratic equations—if they are all satisfied, we should get a 0, and this only requires a single query!

Consider the function M mapping all linear combinations of $P_1, ..., P_m$ to their evaluation at $\mathbf{a}$. There are 2^m such equations, so M has 2^m inputs. We are now simply checking if M is equal to the zero function. M is clearly linear (note that a linear combination of quadratics is equal to simply another linear combination of those same quadratics). Therefore, if $M \neq 0$, they must differ on at least half of their inputs and thus for a randomly sampled linear combination, we reject with probability $\frac{1}{2}$. So we perform 2 such samplings and queries, ultimately rejecting with probability at least $\frac{3}{4}$.

Now, we summarize the proof of completeness and soundness. First, assume we are given a satisfiable instance of QUAD-SAT. Then because a satisfying assignment $\mathbf{a}$ will ensure that each P_i is 0, we can take as proof π the WH encoding of $\mathbf{a}$ and $\mathbf{a} \otimes \mathbf{a}$. The BLR test will never reject a linear function, and neither will the test ensuring that g and f are consistent. We also know that because each P_i evaluated at $\mathbf{a}$ is zero (and because f is actually linear), any linear combination will also equal 0. So, we accept yes instances with probability 1.

Assume we are given some unsatisfiable instance. If the functions f and g are not linear, as explained above, we can reject either with probability $\frac{1}{2}$, repeating the test r times to achieve rejection with probability $1 - 2^{-r}$ for any (constant) r. If f and g are both .99-close to linear but f does not encode the tensor product of the vector encoded by g, we know we reject with probability $\frac{3}{4}$, shown above. If we pass both the linearity and decoding tests, we know the test on the function M as described above will reject with probability at least $\frac{3}{4}$ because any assignment $\mathbf{a}$ will not be able to satisfy at least one linear combination of P_i's (because we have a no instance). So, on no instances, we reject with probability at least $\frac{1}{2}$.

Note that for the BLR test, we perform constantly-many queries (based on the desired closeness of f and g to linearity), $O(1)$ queries for the test that f encodes the tensor product of the vector encoded by g, and $O(1)$ queries to check if the assignment is satisfying. Randomness is only used to sample polynomially-sized vectors, so we use polynomially many random bits.

This concludes the proof that $NP \subseteq PCP(poly(n), O(1))$.

$\square$