CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 15 (Mar. 5): $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly}(n), O(1))$

Lecturer: Zachary Friggstad *Scribe: Zachary Friggstad*

15.1 Preamble

All vectors in this lecture are column vectors unless explicitly indicated otherwise. We associate bits $\{0,1\}$ with integers modulo 2, so for $x, y \in \{0,1\}$ we have $x + y$ is the sum of these bits mod 2 which is the same as the XOR of the bits.

Today, we proved the following:

Theorem 1 $\mathbf{NP} \subseteq \mathbf{PCP}(\textit{poly}(n), O(1))$

A major caveat is that the proof itself has exponential size, but it is still far from obvious how to get this result. Also, if we end up discussing the full proof of the PCP theorem then we need to use the tools developed here.

It suffices to develop such a PCP verifier for some **NP**-complete language for the usual reason. If L is **NP**-complete and $L \in \mathbf{PCP}(\text{poly}(n), O(1))$ then in fact $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly}(n), O(1))$ because we can reduce any $L' \in \mathbf{NP}$ to L in polynomial time and then expect the proof for the verifier to be a proof for the resulting instance of the L.

For most results in this course concerning how **NP** relates to other classes, we focussed on studying SAT. Here, we do something different. We show that QUADEQ $\in \mathbf{PCP}(\text{poly}(n), O(1))$ where QUADEQ is the **NP**-complete problem from the first assignment.

We still recall its definition. In an instance of QUADEQ we have a collection of variables $u_1, \ldots, u_n$ over integers modulo 2 and a collection of m *strict quadratic constraints*, each of the form $\sum_{1 \leq i,j \leq n} a_{i,j} \cdot u_i \cdot u_j = b$ where each $a_{i,j}$ and b are also integers modulo 2. The goal is to determine if there is some setting to the variables that causes all constraints to hold.

For example:

$$\begin{aligned} u_1 \cdot u_1 + u_2 \cdot u_3 &= 1 \\ u_1 \cdot u_2 + u_2 \cdot u_2 + u_3 \cdot u_3 &= 0 \\ u_1 \cdot u_1 + u_2 \cdot u_2 + u_3 \cdot u_3 &= 1 \end{aligned}$$

where we understand the equality to be equality of integers modulo 2. The only solution to this system is $u = (1,0,0)^T$. You proved the general problem QUADEQ was **NP**-complete on the first assignment.

We recast this problem slightly differently. For a vector $u \in \{0,1\}^n$, let $u \otimes u$, the **tensor** of u with itself, be the vector in $\{0,1\}^{n^2}$ that is indexed by pairs (i,j) where $(u \otimes u)_{(i,j)} = u_i \cdot u_j$. It is helpful to visualize it this way. Say $n = 3$, then:

$$u \otimes u = \begin{pmatrix} u_1 \cdot u \\ u_2 \cdot u \\ u_3 \cdot u \end{pmatrix} = \left(\begin{array}{c} u_1 \cdot u_1 \\ u_1 \cdot u_2 \\ u_1 \cdot u_3 \\ \hline u_2 \cdot u_1 \\ u_2 \cdot u_2 \\ u_2 \cdot u_3 \\ \hline u_3 \cdot u_1 \\ u_3 \cdot u_2 \\ u_3 \cdot u_3 \end{array}\right).$$

The horizontal separators are just to help the visualization, the last object is a column vector in $\{0,1\}^9$.

The constraints of a QUADEQ instance can then neatly be presented as $A \cdot (x \otimes x) = b$ where $A \in \{0,1\}^{m \times n^2}$ is the "constraint matrix" and $b \in \{0,1\}^m$ is the vector on the right-hand side of all constraints. For example, the above system could be written as

$$\left(\begin{array}{ccccccccc} (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \cdot (u \otimes u) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

The pairs of indices in the first matrix are just to help visualize our indexing scheme for how pairs of indices correspond to the various columns.

There is some ambiguity on whether we should put a 1 at entry (i,j) or (j,i) for a quadratic term of the form $x_i \cdot x_j$: it does not matter which one we pick.

15.2 Linear Functions and Encodings

Definition 1 *A function $f : \{0,1\}^d \to \{0,1\}$ is* **linear** *if $f(x+y) = f(x) + f(y)$ for each $x, y \in \{0,1\}^n$.*

We do not require this explicitly in the discussion, but it is good to think about how linear functions are really just giving dot products against a fixed vector.

Lemma 1 *A function $f : \{0,1\}^d \to \{0,1\}$ is linear if and only if there is some $u \in \{0,1\}^d$ such that $f(x) = x \circ u$ for each $x \in \{0,1\}^d$ where we let $x \circ u = \sum_{i=1}^d u_i \cdot u_i$ be the standard dot product (mod 2, of course).*

One of the easier exercise questions is to check the details of this claim.

The reason we are studying linear functions is that the PCP verifier for QUADEQ will not expect the proof to supply just the vector u itself that is supposed to satisfy the system. Rather, it expects the *entire* truth table of the function $x \circ u$ over all possible inputs x. This is exponential in the size of u. But, as we will see, this allows us to verify the proof probabilistically.

To get some insight into this, consider the following principle that shows how even a slight difference in vectors can cause their truth-table encodings to differ greatly.

Lemma 2 (Random Subsum Principle) *For $u \neq v$ in $\{0,1\}^d$, $\mathbf{Pr}_{x \sim \{0,1\}^d}[u \circ x \neq v \circ x] = 1/2$.*

Proof. Suppose i is such that $u_i \neq v_i$. Pair up vectors in $\{0,1\}^d$ where x, x' are paired iff they only differ on bit i. Then $u \circ x = v \circ x$ if and only if $x' \circ u \neq x' \circ v$ for such a pair $\{x, x'\}$.

Thus, $u \circ x$ and $v \circ x$ agree on precisely half of the vectors $x \in \{0,1\}^d$. ■

15.2.1 Error Correcting Codes

This is your first introduction to using error correcting codes in complexity theory. Unfortunately we do not have time in this course to explore deeper applications. But let us at least briefly discuss this connection a bit. Think of the various $u \in \{0,1\}^d$ as *messages* to be sent. Let $\texttt{WH}_u \in \{0,1\}^{2^d}$ be the vector consisting of all 2^d values $x \circ u$ for the various $x \in \{0,1\}^d$ (the notation comes from "Walsh-Hadamard").

The vector $\texttt{WH}_u$ is an "encoding" of u using a tremendous number of extra bits. But the random subsum principle indicates that if $u \neq v$, even in just a single bit, then $\texttt{WH}_u$ and $\texttt{WH}_v$ differ in half of their coordinates. Furthermore, if $y \in \{0,1\}^{2^d}$ is supposed to represent some $\texttt{WH}_u$ but differs in less than 1/4 of its coordinates, we can still uniquely determine which u that y was supposed to represent. Look for how this idea is being used implicitly in the proof below.

We also think of $\texttt{WH}_u$ as a function over $\{0,1\}^d$ via $\texttt{WH}_u(x) = u \circ x$. Note that $\texttt{WH}_u$ is a linear function and every linear function over d bits is of the form $\texttt{WH}_v$ for some v.

15.3 Verifying a Proof for QUADEQ

The verifier for a QUADEQ instance with n variables and m constraints in the form $A \cdot (u \otimes u) = b$ expects the proof string π to have length $2^n + 2^{n^2}$. The first 2^n bits are to be interpreted as the 2^n values of some function $f : \{0,1\}^n \to \{0,1\}$ and the last 2^{n^2} bits are interpreted as a function $g : \{0,1\}^{n^2} \to \{0,1\}$.

Ideally, the proof string will encode the linear functions $f = \texttt{WH}_u$ and $g = \texttt{WH}_{u \otimes u}$ where u satisfies the QUADEQ system. But, of course, the proof may "cheat" in many forms: it may not use a satisfying u, g may encode a different linear function $\texttt{WH}_v$ for $v \in \{0,1\}^{n^2}$, or the functions themselves might not even be linear!

We describe how to verify a proof with increasing degrees of skepticism.

15.3.1 Assuming $f = \texttt{WH}_u$ and $g = \texttt{WH}_{u \otimes u}$ for some $u \in \{0,1\}^n$

That is, we suppose that f and g correctly encode linear functions over u and its tensor $u \otimes u$. But does u actually satisfy the system?

We show how to query the proof so that if u does satisfy the QUADEQ instance then the verifier always accepts, but if it does not then the verifier will probably reject (with large constant probability). This might seem challenging: perhaps u does not satisfy only one constraint. How can we determine this with only a constant number of queries to the proof string?

The idea is that if we have even a single unsatisfied constraint (under u), then with probability 1/2 summing a random subset of constraints yields a single constraint that is still not satisfied: the random subsum principle!

Verifier
Repeat 10 times:

- Sample $r \sim \{0,1\}^m$.

- Compute $y = r^T \cdot A$ (i.e. sum the rows of A that correspond to indices i with $r_i = 1$).
- If $g(y^T) \neq r^T \cdot b$, `reject`

Observe the proof string is only queried 10 times.

Now, the query $g(y^T)$ is the same as $r^T \cdot A \cdot (u \otimes u)$ and the check is asking if $r^T \cdot A \cdot (u \otimes u) = r^T \cdot b$. If u satisfies the system, clearly the verifier does not reject. Otherwise,

Lemma 3 *If $A \cdot (u \otimes u) \neq b$, then the verifier rejects with probability at least $1 - (1/2)^{10}$.*

Proof. Consider a single iteration. Here the vectors $A \cdot (u \otimes u)$ and b are distinct. So by the random subsum principle, $\mathbf{Pr}_{r\sim\{0,1\}^m}[r^T \cdot A \cdot (u \otimes u) \neq r^T \cdot b] = 1/2$. But $g(y^T) = r^T \cdot A \cdot (u \otimes u)$ by the assumption $g = \mathtt{WH}_{u\otimes u}$, so $\mathbf{Pr}_{r\sim\{0,1\}^m}[g(y^T) \neq r^T \cdot b] = 1/2$. ■

This only works under the assumption that f and g are linear and g is the Walsh-Hadamard code for the tensor $u \otimes u$ of the vector u encoded by f. Next, we lift the latter assumption.

15.3.2 Assuming only that f and g are linear

Suppose $f = \mathtt{WH}_u$ and $g = \mathtt{WH}_v$ for some $u \in \{0,1\}^n$ and $v \in \{0,1\}^{n^2}$. We describe a test that will probably reject if $v \neq u \otimes u$.

Verifier
Repeat 10 times:

- Sample $r, r' \sim \{0,1\}^n$ independently.
- Compute[1] $r \otimes r'$.
- If $f(r) \cdot f(r') \neq g(r \otimes r')$, `reject`.

Observe the proof string is only queried 30 times.

To see why this works, we rephrase how the tests work. Consider the following two matrices:

- $U \in \{0,1\}^n$ where $U_{i,j} = u_i \cdot u_j$.
- $V \in \{0,1\}^n$ where $V_{i,j} = v_{(i,j)}$, recalling our convention that $\{0,1\}^{n^2}$ is indexed by pairs of indices between 1 and n. So, V is really the same as v except it is arranged as a matrix rather than a flattened-out column vector.

Lemma 4 *For any $r, r' \in \{0,1\}^n$, $f(r) \cdot f(r') = r^T \cdot U \cdot r'$ and $g(r \otimes r') = r^T \cdot V \cdot r'$.*

Proof. By assumption $f = \mathtt{WH}_u$,

$$f(r) \cdot f(r') = (r \circ u) \cdot (r' \circ u) = \sum_{i,j} r_i \cdot u_i \cdot u_j \cdot r'_j = r^T \cdot U \cdot r'.$$

[1]Coordinate (i,j) of $r \otimes r'$ is $r_i \cdot r'_j$

The other claim is very similar,

$$g(r \otimes r') = (r \otimes r') \circ v = \sum_{(i,j)} (r \otimes r')_{(i,j)} \cdot v_{(i,j)} = \sum_{i,j} r_i \cdot r'_j \cdot V_{i,j} = r^T \cdot V \cdot r'.$$

■

With this view, if $v = u \otimes u$ then $V = U$ so the test never rejects. Otherwise,

Lemma 5 *If $v \neq u \otimes u$, the test rejects with probability at least $1 - (3/4)^{10}$.*

Proof. Again, consider a single iteration of the verification. If $v \neq u \otimes u$ then $V \neq U$. Suppose column i of V and U differ. Then by the random subsum principle, entry i of $r^T \cdot U$ and entry i of $r^T \cdot V$ differ with probability 1/2. Thus,

$$\mathbf{Pr}_{r \sim \{0,1\}^n}[r^T \cdot U \neq r^T \cdot V] \geq 1/2.$$

In the event $r^T \cdot U \neq r^T \cdot V$, then the random subsum principle *again* shows

$$\mathbf{Pr}_{r' \sim \{0,1\}^n}[r^T \cdot U \cdot r' \neq r^T \cdot V \cdot r' | r^T \cdot U \neq r^T \cdot V] = 1/2.$$

So the probability that both $r^T \cdot U \neq r^T \cdot V$ and, then, $r^T \cdot U \cdot r' \neq r^T \cdot V \cdot r'$ is at least 1/4. ■

The remaining assumption we still have to lift is that f and g are indeed linear functions.

15.3.3 Verifying f and g are (nearly) linear

The test is very simple!

Verifier
Repeat 2000 times:

- Sample $x, y \sim \{0,1\}^n$.
- If $f(x+y) \neq f(x) + f(y)$, `reject`.

Similarly, check $g(x+y) = g(x) + g(y)$ with 2000 independent samples of pairs $x, y \sim \{0,1\}^{n^2}$. In total, this makes 12000 queries.

If f and g are linear, clearly these tests always pass. But what if they are not? Unfortunately we cannot reject with great probability if they are very close to linear. This test only rejects those that are somewhat "far" from being linear.

The following is not obvious, we may see a proof later in the course. For now, we take it at face value. Think of it as a *quantitative* generalization of the fact that truly linear functions f are of the form $\mathtt{WH}_u$ for some vector u.

Theorem 2 *Suppose $f : \{0,1\}^d \to \{0,1\}$ is such that $\mathbf{Pr}_{x,y \sim \{0,1\}^d}[f(x+y) = f(x) + f(y)] \geq 1 - \delta$ for some $\delta < 1/4$. Then there is a unique $u \in \{0,1\}^d$ such that $\mathbf{Pr}_{x \sim \{0,1\}^d}[f(x) = u \circ x] \geq 1 - \delta$.*

That u is unique is because if $u \neq v$ then $u \circ x \neq v \circ x$ for half of the different $x \in \{0,1\}^d$. The existence of such u is far less obvious.

Definition 2 *Say that a function is δ-close to being linear if there is some u such that $\mathbf{Pr}_{x \sim \{0,1\}^d}[f(x) = u \circ x] \geq 1 - \delta$.*

Theorem 2 immediately yields the following.

Corollary 1 *If either f or g is not 0.001-close to being linear, then the above test rejects with probability at least* $1 - 0.999^{2000} \geq 0.85$.

15.3.4 Local Decoding

The last tests we described did not reject f or g with good enough probability if they were not linear. Only if they were somewhat far from being linear.

Suppose f is 0.001-close to $\mathtt{WH}_u$ and g is 0.001-close to $\mathtt{WH}_v$ for some $u \in \{0,1\}^n$ and $v \in \{0,1\}^{n^2}$. We would like to continue the other tests described earlier with these linear functions. But we only have access to the "almost linear" functions f and g.

We use one last trick: local decoding. If we know, say, f is close to being linear then we can in fact get $\mathtt{WH}_u$ for *any* input x with good probability, even those x for which $f(x) \neq u \circ x$.

Lemma 6 (Local Decoding) *Suppose f is δ-close to a linear function $x \circ u$. The for any $x \in \{0,1\}^n$,* $\mathbf{Pr}_{y \sim \{0,1\}^n}[f(y) + f(x+y) = x \circ u] \geq 1 - 2\delta$.

Proof. By the union bound, the probability that at least one of of $f(y)$ or $f(x+y)$ does not agree with $\mathtt{WH}_u(y)$ or $\mathtt{WH}_u(x+y)$ (respectively) is at most 2δ. If both agree, we have

$$f(y) + f(x+y) = \mathtt{WH}_u(y) + \mathtt{WH}_u(x+y) = y \circ u + (x+y) \circ u = (x+y+y) \circ u = x \circ u.$$

Recalling $y + y$ is the zero vector when working with integers modulo 2. ■

Moral: If we want $x \circ u$ where u is the unique vector such that f is δ-close to $\mathtt{WH}_u$, then we can get this value with probability at least $1 - 2\delta$ by only making 2 queries to f by choosing y randomly and returning $f(y) + f(x+y)$.

Whenever we say to compute a value of f, say $f(x)$, **using local decoding** we really mean to compute $f(y) + f(x+y)$ for a randomly chosen y.

15.4 Putting It All Together

The verifier runs the following tests:

1) For 2000 iterations:

- Test $f(x+y) = f(x) + f(y)$ for random $x, y \in \{0,1\}^n$, rejecting if this fails.
- Test $g(x+y) = g(x) + g(y)$ for random $x, y \in \{0,1\}^n$, rejecting if this fails.

2) For 10 iterations:

- Test $f(r) \cdot f(r') = g(r \otimes r')$ using local decoding (on both f and g) for random $r, r' \in \{0,1\}^n$, rejecting if this fails.

3) For 10 iterations:

- Test $g(y^T) = r^T \cdot b$ using local decoding where $y = r^T \cdot A$ for some random $r \in \{0,1\}^m$, rejecting if this fails.

Accept if none of the tests rejected.

Note that no test depends on any other so they may all be performed "in parallel". That is, all different indices into the proof (i.e. inputs to f and g) can be sampled before any of them are queried. Considering that local decoding performs two queries each time it is invoked, this procedure makes the following number of queries:

$$2000 \cdot 6 + 10 \cdot 3 \cdot 2 + 10 \cdot 2 = 12080.$$

If $f = \mathtt{WH}_u$ and $g = \mathtt{WH}_{u \circ u}$ where u satisfies the `QUADEQ` instance then, as discussed, all tests will pass.

Lemma 7 *If the `QUADEQ` instance is not satisfiable, then for any functions f, g (i.e. any proof string) the verifier will reject with probability at least 1/2.*

Proof. We consider some cases, where each case assumes the previous case(s) did not apply:

- **Case**: One of f or g is not 0.001-close to being linear.

  Then the verifier rejects with probability at least $1 - 0.999^{2000} \geq 1/2$.

- **Case**: f is 0.001-close to $\mathtt{WH}_u$ and g is 0.001-close to $\mathtt{WH}_v$ where $v \neq u \otimes u$.

  With probability at least $1-(3/4)^{10}$ there is some sampled pair (r, r') such that $\mathtt{WH}_u(r) \cdot \mathtt{WH}_u(r') \neq \mathtt{WH}_v(r \otimes r')$.

  There are 30 queries in total, so all local decoding steps succeed with probability at least $1-60 \cdot 0.001 = 0.94$ no matter which r or r' are sampled. So with probability at least $1 - (0.1 + 0.06) \geq 0.5$, the verifier would reject.

- **Case**: The function $\mathtt{WH}_{u \otimes u}$ that g is 0.001-close to does not have u satisfying the `QUADEQ` instance.

  In this case, the probability that some sampled r has $\mathtt{WH}_{u \otimes u}(y^T) \neq r^T \cdot b$ where $y = r^T \cdot A$ is at least $1 - 1/2^{10}$. There are 10 queries in total, so all local decoding steps succeed with probability at least $1 - 10 \cdot 0.001$. So with probability at least $1 - (1/2^{10} + 10 \cdot 0.001) \geq 0.5$ the verifier rejects.

Observe that in the case that `QUADEQ` is not satisfiable, any proof string / pair of functions f, g would fall under one of these cases. ■

To wrap this up, note that a polynomial number of random bits are sampled for each query and the verifier runs in polynomial time (assuming it has random access to the proof string). That is,

$$\text{QUADEQ} \in \mathbf{PCP}(\text{poly}(n), O(1)).$$