

11.5 $\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$: PCP FROM THE WALSH-HADAMARD CODE

We now prove a weaker version of the **PCP** Theorem, showing that every **NP** statement has an exponentially long proof that can be locally tested by only looking at a constant number of bits. In addition to giving a taste of how one proves **PCP** theorems, techniques from this section will be used in the proof of the full-fledged **PCP** Theorem in Chapter 22.

Theorem 11.19 (Exponential-sized **PCP** system for **NP** [ALM⁺92]) $\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$.

We prove this theorem by designing an appropriate verifier for an **NP**-complete language. The verifier expects the proof to contain an encoded version of the usual certificate. The verifier checks such an encoded certificate by simple probabilistic tests.

11.5.1 Tool: Linearity testing and the Walsh-Hadamard code

We use the *Walsh-Hadamard code* (see also Section 19.2.2, though the treatment here is self-contained). It is a way to encode bit strings of length n by *linear functions* in n variables over $\text{GF}(2)$. The encoding function $\text{WH} : \{0, 1\}^n \rightarrow \{0, 1\}^{2^n}$ maps a string $\mathbf{u} \in \{0, 1\}^n$ to the truth table of the function $\mathbf{x} \mapsto \mathbf{u} \odot \mathbf{x}$, where for $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ we define $\mathbf{x} \odot \mathbf{y} = \sum_{i=1}^n x_i y_i \pmod{2}$. Note that this is a very inefficient encoding method: An n -bit string $\mathbf{u} \in \{0, 1\}^n$ is encoded using $|\text{WH}(\mathbf{u})| = 2^n$ bits. If $f \in \{0, 1\}^{2^n}$ is equal to $\text{WH}(\mathbf{u})$ for some $\mathbf{u}$, then we say that f is a *Walsh-Hadamard codeword*. Such a string $f \in \{0, 1\}^{2^n}$ can also be viewed as a function from $\{0, 1\}^n$ to $\{0, 1\}$.

Below, we repeatedly use the following fact (see Claim A.31).

RANDOM SUBSUM PRINCIPLE: If $\mathbf{u} \neq \mathbf{v}$ then for $1/2$ the choices of $\mathbf{x}$, $\mathbf{u} \odot \mathbf{x} \neq \mathbf{v} \odot \mathbf{x}$.

The random subsum principle implies that the Walsh-Hadamard code is an error-correcting code with minimum distance $1/2$, by which we mean that for every $\mathbf{u} \neq \mathbf{v} \in \{0, 1\}^n$, the encodings $\text{WH}(\mathbf{u})$ and $\text{WH}(\mathbf{v})$ differ in at least half the bits. Now we talk about local tests for the Walsh-Hadamard code (i.e., tests making only $O(1)$ queries).

Local testing of Walsh-Hadamard code

Suppose we are given access to a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and want to test whether or not f is actually a codeword of Walsh-Hadamard. Since the Walsh-Hadamard codewords are precisely the set of all *linear functions* from $\{0, 1\}^n$ to $\{0, 1\}$, we can test f by checking that

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \quad (11.3)$$

for all the 2^{2n} pairs $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ (where “+” on the left side of (11.3) denotes vector addition over $\text{GF}(2)^n$ and on the right side denotes addition over $\text{GF}(2)$). This test works by definition, but it involves reading all 2^n values of f .

Can we test f by reading only a *constant* number of its values? The natural test is to choose $\mathbf{x}, \mathbf{y}$ at random and verify (11.3). Clearly, even such a local test accepts

a linear function with probability 1. However, now we can no longer guarantee that every function that is not linear is rejected with high probability! For example, if f is very close to being a linear function, meaning that f is obtained by modifying a linear function on a very small fraction of its inputs, then such a *local* test will encounter the nonlinear part with very low probability, and thus not be able to distinguish f from a linear function. So we set our goal less ambitiously: a test that on one hand accepts every linear function, and on the other hand rejects with high probability every function that is *far from linear*. The natural test suffices for this job.

Definition 11.20 Let $\rho \in [0, 1]$. We say that $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$ are ρ -close if $\Pr_{\mathbf{x} \in_{\mathcal{R}} \{0, 1\}^n} [f(\mathbf{x}) = g(\mathbf{x})] \geq \rho$. We say that f is ρ -close to a linear function if there exists a linear function g such that f and g are ρ -close. $\diamond$

Theorem 11.21 (*Linearity Testing* [BLR90]) Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be such that

$$\Pr_{\mathbf{x}, \mathbf{y} \in_{\mathcal{R}} \{0, 1\}^n} [f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})] \geq \rho$$

for some $\rho > 1/2$. Then f is ρ -close to a linear function. $\diamond$

We defer the proof of Theorem 11.21 to Section 22.5 of Chapter 22. For every $\delta \in (0, 1/2)$, we can obtain a linearity test that rejects with probability at least $1/2$ every function that is not $(1 - \delta)$ -close to a linear function, by testing Condition (11.3) repeatedly $O(1/\delta)$ times with independent randomness. We call such a test a $(1 - \delta)$ -linearity test.

Local decoding of Walsh-Hadamard code

Suppose that for $\delta < \frac{1}{4}$ the function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is $(1 - \delta)$ -close to some linear function $\tilde{f}$. Because every two linear functions differ on half of their inputs, the function $\tilde{f}$ is uniquely determined by f . Suppose we are given $\mathbf{x} \in \{0, 1\}^n$ and random access to f . Can we obtain the value $\tilde{f}(\mathbf{x})$ using only a constant number of queries? The naive answer is that since most $\mathbf{x}$'s satisfy $f(\mathbf{x}) = \tilde{f}(\mathbf{x})$, we should be able to learn $\tilde{f}(\mathbf{x})$ with good probability by making only the single query $\mathbf{x}$ to f . The problem is that $\mathbf{x}$ could very well be one of the places where f and $\tilde{f}$ differ. Fortunately, there is still a simple way to learn $\tilde{f}(\mathbf{x})$ while making only two queries to f :

1. Choose $\mathbf{x}' \in_{\mathcal{R}} \{0, 1\}^n$.
2. Set $\mathbf{x}'' = \mathbf{x} + \mathbf{x}'$.
3. Let $\mathbf{y}' = f(\mathbf{x}')$ and $\mathbf{y}'' = f(\mathbf{x}'')$.
4. Output $\mathbf{y}' + \mathbf{y}''$.

Since both $\mathbf{x}'$ and $\mathbf{x}''$ are individually uniformly distributed (even though they are dependent), by the union bound with probability at least $1 - 2\delta$ we have $\mathbf{y}' = \tilde{f}(\mathbf{x}')$ and $\mathbf{y}'' = \tilde{f}(\mathbf{x}'')$. Yet by the linearity of $\tilde{f}$, $\tilde{f}(\mathbf{x}) = \tilde{f}(\mathbf{x}' + \mathbf{x}'') = \tilde{f}(\mathbf{x}') + \tilde{f}(\mathbf{x}'')$, and hence with at least $1 - 2\delta$ probability, $\tilde{f}(\mathbf{x}) = \mathbf{y}' + \mathbf{y}''$. (We use here the fact that over $\text{GF}(2)$, $a + b = a - b$.) This technique is called *local decoding* of the Walsh-Hadamard code since it allows to recover any bit of the correct codeword (the linear function $\tilde{f}$) from a corrupted version (the function f) while making only a constant number of queries. It is also known as *self correction* of the Walsh-Hadamard code.



Figure 11.2. The PCP proof that a set of quadratic equations is satisfiable consists of $\text{WH}(u)$ and $\text{WH}(u \otimes u)$ for some vector u . The verifier first checks that the proof is close to having this form and then uses the local decoder of the Walsh-Hadamard code to ensure that u is a solution for the quadratic equation instance. The dotted areas represent corrupted coordinates.

11.5.2 Proof of Theorem 11.19

We will show a $(\text{poly}(n), 1)$ -verifier proof system for a particular **NP**-complete language L . The result that $\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$ follows since every **NP** language is reducible to L . The **NP**-complete language L we use is **QUADEQ**, the language of systems of quadratic equations over $\text{GF}(2) = \{0, 1\}$ that are satisfiable.

EXAMPLE 11.22

The following is an instance of **QUADEQ** over the variables $u_1, \dots, u_5$:

$$u_1 u_2 + u_3 u_4 + u_1 u_5 = 1$$

$$u_2 u_3 + u_1 u_4 = 0$$

$$u_1 u_4 + u_3 u_5 + u_3 u_4 = 1$$

This instance is satisfiable since the all-1 assignment satisfies all the equations.

QUADEQ is **NP**-complete, as can be checked by reducing from the **NP**-complete language **CKT-SAT** of satisfiable Boolean circuits (see Section 6.1.2). The idea is to have a variable represent the value of each wire in the circuit (including the input wires) and to express **AND** and **OR** using the equivalent quadratic polynomial: $x \vee y = 1$ iff $(1-x)(1-y) = 0$, and so on. Details are left as Exercise 11.15.

Since $u_i = (u_i)^2$ in $\text{GF}(2)$, we can assume the equations do not contain terms of the form u_i (i.e., all terms are of degree exactly two). Hence m quadratic equations over the variables $u_1, \dots, u_n$ can be described by an $m \times n^2$ matrix A and an m -dimensional vector $\mathbf{b}$ (both over $\text{GF}(2)$). Indeed, the problem **QUADEQ** can be phrased as the task, given such $A, \mathbf{b}$, of finding an n^2 -dimensional vector U satisfying (1) $AU = \mathbf{b}$ and (2) U is the tensor product $\mathbf{u} \otimes \mathbf{u}$ of some n -dimensional vector $\mathbf{u}$.⁴

The PCP verifier

We now describe the **PCP** system for **QUADEQ** (see Figure 11.2). Let $A, \mathbf{b}$ be an instance of **QUADEQ** and suppose that $A, \mathbf{b}$ is satisfiable by an assignment $\mathbf{u} \in \{0, 1\}^n$. The verifier V gets access to a proof $\pi \in \{0, 1\}^{2n^2}$, which we interpret as a pair of functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and $g : \{0, 1\}^{n^2} \rightarrow \{0, 1\}$. In the correct **PCP** proof π for $A, \mathbf{b}$, the

⁴ If $\mathbf{x}, \mathbf{y}$ are two n -dimensional vectors then their tensor product, denoted $\mathbf{x} \otimes \mathbf{y}$, is the n^2 -dimensional vector (or $n \times n$ matrix) whose (i, j) th entry is $x_i y_j$ (identifying $[n^2]$ with $[n] \times [n]$ in some canonical way). See also Section 21.3.3.

function f will be the Walsh-Hadamard encoding for $\mathbf{u}$, and the function g will be the Walsh-Hadamard encoding for $\mathbf{u} \otimes \mathbf{u}$. That is, we will design the **PCP** verifier V in a way ensuring that it accepts proofs of this form with probability one, hence satisfying the completeness condition. The analysis repeatedly uses the random subsum principle.

Step 1: Check that f, g are linear functions. As already noted, this isn't something that the verifier can check per se using local tests. Instead, the verifier performs a 0.999-linearity test on both f, g , and rejects the proof at once if either test fails.

Thus, if either of f, g is not 0.999-close to a linear function, then V rejects with high probability. Therefore for the rest of the procedure we can assume that there exist two linear functions $\tilde{f} : \{0, 1\}^n \rightarrow \{0, 1\}$ and $\tilde{g} : \{0, 1\}^{n^2} \rightarrow \{0, 1\}$ such that $\tilde{f}$ is 0.999-close to f , and $\tilde{g}$ is 0.999-close to g . (Note: In a correct proof, the tests succeed with probability 1 and $\tilde{f} = f$ and $\tilde{g} = g$.)

In fact, we will assume that for Steps 2 and 3, the verifier can query $\tilde{f}, \tilde{g}$ at any desired point. The reason is that local decoding allows the verifier to recover any desired value of $\tilde{f}, \tilde{g}$ with good probability, and Steps 2 and 3 will only use a small (less than 20) number of queries to $\tilde{f}, \tilde{g}$. Thus with high probability (say > 0.9) local decoding will succeed on all these queries.

NOTATION: To simplify notation and in the rest of the procedure we use f, g for $\tilde{f}, \tilde{g}$ respectively. (This is OK since as argued previously, V can query $\tilde{f}, \tilde{g}$ at will.) In particular we assume both f and g are linear, and thus they must encode some strings $\mathbf{u} \in \{0, 1\}^n$ and $\mathbf{w} \in \{0, 1\}^{n^2}$. In other words, f, g are the functions given by $f(\mathbf{r}) = \mathbf{u} \odot \mathbf{r}$ and $g(\mathbf{z}) = \mathbf{w} \odot \mathbf{z}$.

Step 2: Verify that g encodes $\mathbf{u} \otimes \mathbf{u}$, where $\mathbf{u} \in \{0, 1\}^n$ is the string encoded by f . The verifier does the following test ten times using independent random bits: "Choose $\mathbf{r}, \mathbf{r}'$ independently at random from $\{0, 1\}^n$, and if $f(\mathbf{r})f(\mathbf{r}') \neq g(\mathbf{r} \otimes \mathbf{r}')$ then halt and reject."

In a correct proof, $\mathbf{w} = \mathbf{u} \otimes \mathbf{u}$, so

$$f(\mathbf{r})f(\mathbf{r}') = \left(\sum_{i \in [n]} u_i r_i \right) \left(\sum_{j \in [n]} u_j r'_j \right) = \sum_{i,j \in [n]} u_i u_j r_i r'_j = (\mathbf{u} \otimes \mathbf{u}) \odot (\mathbf{r} \otimes \mathbf{r}'),$$

which in the correct proof is equal to $g(\mathbf{r} \otimes \mathbf{r}')$. Thus Step 2 never rejects a correct proof.

Suppose now that, unlike the case of the correct proof, $\mathbf{w} \neq \mathbf{u} \otimes \mathbf{u}$. We claim that in each of the ten trials V will halt and reject with probability at least $\frac{1}{4}$. (Thus the probability of rejecting in at least one trial is at least $1 - (3/4)^{10} > 0.9$.) Indeed, let W be an $n \times n$ matrix with the same entries as $\mathbf{w}$, let U be the $n \times n$ matrix such that $U_{i,j} = u_i u_j$, and think of $\mathbf{r}$ as a row vector and $\mathbf{r}'$ as a column vector. In this notation,

$$g(\mathbf{r} \otimes \mathbf{r}') = \mathbf{w} \odot (\mathbf{r} \otimes \mathbf{r}') = \sum_{i,j \in [n]} w_{i,j} r_i r'_j = \mathbf{r} W \mathbf{r}'$$

$$f(\mathbf{r})f(\mathbf{r}') = (\mathbf{u} \odot \mathbf{r})(\mathbf{u} \odot \mathbf{r}') = \left(\sum_{i=1}^n u_i r_i \right) \left(\sum_{j=1}^n u_j r'_j \right) = \sum_{i,j \in [n]} u_i u_j r_i r'_j = \mathbf{r} U \mathbf{r}'$$

And V rejects if $\mathbf{r} W \mathbf{r}' \neq \mathbf{r} U \mathbf{r}'$. The random subsum principle implies that if $W \neq U$ then at least $1/2$ of all $\mathbf{r}$ satisfy $\mathbf{r} W \neq \mathbf{r} U$. Applying the random subsum principle for

each such $\mathbf{r}$, we conclude that at least $1/2$ the $\mathbf{r}'$ satisfy $\mathbf{r}W\mathbf{r}' \neq \mathbf{r}U\mathbf{r}'$. We conclude that the trial rejects for at least $1/4$ of all pairs $\mathbf{r}, \mathbf{r}'$.

Step 3: Verify that g encodes a satisfying assignment. Using all that has been verified about f, g in the previous two steps, it is easy to check that any particular equation, say the k th equation of the input, is satisfied by $\mathbf{u}$, namely,

$$\sum_{i,j} A_{k,(i,j)} u_i u_j = b_k \quad (11.4)$$

Denoting by $\mathbf{z}$ the n^2 dimensional vector $(A_{k,(i,j)})$ (where i, j vary over $[1 \dots n]$), we see that the left-hand side is nothing but $g(\mathbf{z})$. Since the verifier knows $A_{k,(i,j)}$ and b_k , it simply queries g at $\mathbf{z}$ and checks that $g(\mathbf{z}) = b_k$.

The drawback of this idea is that in order to check that $\mathbf{u}$ satisfies the entire system, the verifier needs to make a query to g for each $k = 1, 2, \dots, m$, whereas the number of queries is required to be independent of m . Luckily, we can use the random subsum principle again! The verifier takes a random subset of the equations and computes their sum mod 2. (In other words, for $k = 1, 2, \dots, m$ multiply the equation in (11.4) by a random bit and take the sum.) This sum is a new quadratic equation, and the random subsum principle implies that if $\mathbf{u}$ does not satisfy even one equation in the original system, then with probability at least $1/2$ it will not satisfy this new equation. The verifier checks that $\mathbf{u}$ satisfies this new equation.

Overall, we get a verifier V such that (1) if $A, \mathbf{b}$ is satisfiable then V accepts the correct proof with probability 1 and (2) if $A, \mathbf{b}$ is not satisfiable then V accepts every proof with probability at most 0.8. The probability of accepting a proof for a false statement can be reduced to $1/2$ by simple repetition, completing the proof of Theorem 11.19. ■

WHAT HAVE WE LEARNED?

- Computing approximate solutions to **NP**-hard problems is an important research endeavor. The classical Cook-Levin-Karp reductions did not rule out the existence of approximation algorithms for many interesting **NP**-hard problem.
- Nontrivial approximation algorithms can be designed for many **NP**-hard problems.
- The **PCP** Theorem gives a new probabilistic characterization of **NP**, and also shows that the MAX-3SAT problem cannot be approximated within arbitrary accuracy if $\mathbf{P} \neq \mathbf{NP}$. In fact, these two results are equivalent to one another.
- There are many other **PCP** theorems with different choices of parameters. In this chapter we saw a simple one where the verifier uses $\text{poly}(n)$ random bits and examines only $O(1)$ bits in the proof.
- Proofs of **PCP** theorems involve some interesting way to encode a satisfying assignment to a boolean formula. This is accompanied by procedures that can efficiently check any string that is claimed to be such an encoding. In this chapter we saw a proof using the Hadamard code (that just consists of linear functions over $\text{GF}(2)$).