Penn State University
CSE 564 Complexity of Combinatorial Problems

Instructor: Pei Wu

Module
6

PCP (i): hardness of approximation

In the previous lectures we developed interactive proofs and algebraic verification tools such as arithmetization and the sum-check protocol. This lecture begins our study of *probabilistically checkable proofs* (PCPs), a remarkable way to represent proofs so that a probabilistic verifier can spot-check only $O(1)$ locations of a (possibly long) proof and still be confident in its correctness.

The headline result in this area is the PCP theorem, which (in one standard form) states that every NP language has proofs that can be checked using only $O(\log n)$ random bits and $O(1)$ queries. Beyond its conceptual impact, the PCP theorem has a major algorithmic consequence: it yields *hardness of approximation* results for many optimization problems, showing that even getting close to the optimum is NP-hard.

We first define PCP verifiers and record some basic sanity checks. We then discuss the connection between PCPs and approximation problems such as Max-3SAT, Max-Cut, and Max-3XOR, including Håstad's tight inapproximability thresholds. Finally, we sketch a warm-up "baby PCP" for an NP-complete quadratic feasibility problem using the Hadamard code and three core testing ideas: a linearity test (BLR), a tensor/consistency test, and a global feasibility test.

6.1 PCP verifiers

In the usual NP picture, the prover sends a proof π and the verifier reads the whole proof deterministically. In a PCP, the verifier is randomized and treats the proof as an *oracle* that can be queried at a few locations.

Definition 6.1 (PCP verifier). Fix an input length n. A *PCP verifier* for a language $L \subseteq \{0,1\}^*$ is a probabilistic polynomial-time machine V that, on input $x \in \{0,1\}^n$, has oracle access to a proof string $\pi \in \Sigma^N$ over some alphabet Σ. The verifier uses $R(n)$ random bits and makes at most $q(n)$ oracle queries. After seeing the answers, V outputs *accept* or *reject*.

Definition 6.2 ($\mathrm{PCP}(R, q, \Sigma, c, s)$). For functions $R(n)$, $q(n)$, an alphabet Σ, and parameters $c > s$ in $[0,1]$, we say $L \in \mathrm{PCP}(R, q, \Sigma, c, s)$ if there exists a PCP verifier V such that:

- **Completeness:** If $x \in L$, then there exists a proof π with $\Pr[V^\pi(x) \text{ accepts}] \geq c$.
- **Soundness:** If $x \notin L$, then for every proof π, $\Pr[V^\pi(x) \text{ accepts}] \leq s$.

The probability is over the verifier's random coins.

The notes use the shorthand $\text{PCP}(R, q, \Sigma, 2/3, 1/3)$ and annotate R as the number of random bits and q as the number of queries.

Remark 6.3 (proof length). If a verifier uses R random bits and q queries, then there are only 2^R possible random strings, and thus only $q \cdot 2^R$ proof locations are ever queried. Therefore, the proof length can be assumed (without loss of generality) to be at most $q \cdot 2^R$. In particular, when $R = O(\log n)$ and $q = O(1)$, the proof length is polynomial.

6.2 Sanity checks and easy containments

The definition interpolates between several familiar complexity classes. Let us write $\text{PCP}(R, q)$ informally when the alphabet and constant completeness/soundness are understood.

- $\text{PCP}(0, 0) = \mathsf{P}$: the verifier is deterministic and never reads the proof, so acceptance depends only on x.
- $\text{PCP}(\log n, 0) = \mathsf{P}$: with $O(\log n)$ random bits and no proof queries, the verifier has only $\text{poly}(n)$ possible random strings, so we can deterministically enumerate them and compute the exact acceptance probability.
- $\text{PCP}(\text{poly}(n), 0) = \mathsf{BPP}$: if the verifier makes no proof queries, the prover is irrelevant and we simply have a randomized polynomial-time algorithm.
- $\text{PCP}(0, \text{poly}(n)) = \mathsf{NP}$: a deterministic verifier can read polynomially many bits of π, which is equivalent to the usual NP verifier that reads the whole witness.
- $\text{PCP}(\log n, O(1)) \subseteq \mathsf{NP}$: with only $O(\log n)$ random bits, there are $\text{poly}(n)$ possible random strings; across all of them, only $O(1) \cdot 2^{O(\log n)} = \text{poly}(n)$ proof positions are ever queried. An NP witness can specify the answers at all those positions, and the verifier can check that *all* random strings lead to acceptance with probability at least 2/3.

6.3 The PCP theorem and an exponential-size warm-up

Theorem 6.4 (PCP theorem (statement from the notes)).

$$\text{PCP}(O(\log n), O(1)) = \mathsf{NP}.$$

The notes mention several landmark proofs (Arora–Safra; Arora–Lund–Motwani–Sudan–Szemerédi) and Dinur's 2005 proof. We will not prove Theorem 6.4 in this lecture; instead we focus on why it is connected to hardness of approximation and then build a weaker "baby PCP" that already illustrates the key coding-theoretic ideas.

Example 6.5 (exponential-size PCP from an AM protocol). The notes give GNI (graph non-isomorphism) as an example of a language with an (inefficient) PCP. Recall the standard AM protocol: Arthur chooses a random graph $\widetilde{G}$ that is a random relabeling of either G_0 or G_1, and Merlin replies with a bit $b \in \{0, 1\}$ indicating which one.

We can "flatten" this into a PCP by letting the proof π be a huge table indexed by all possible Arthur messages (equivalently, by all graphs on n vertices), and storing Merlin's would-be response for each. The PCP verifier samples Arthur's randomness r, determines the corresponding $\widetilde{G}$, and queries the single location $\pi[\widetilde{G}]$ to get Merlin's answer.

This yields a constant-query PCP that uses $\text{poly}(n)$ randomness, but the proof length is exponential (there are exponentially many possible graphs). The PCP theorem can be viewed as showing that for *all* of NP one can achieve constant-query checking with only $O(\log n)$ randomness and hence polynomial proof length.

6.4 Hardness of approximation

We now switch to optimization problems. For a maximization problem, let $\text{OPT}(I)$ denote the value of the best solution for an instance I. An algorithm is an α-approximation if it always outputs a feasible solution of value at least $\alpha \cdot \text{OPT}(I)$.

Definition 6.6 (Max-3SAT). An instance is a 3CNF formula φ with m clauses. The objective is to find an assignment that satisfies as many clauses as possible. Let $\text{OPT}(\varphi)$ be the maximum number of satisfiable clauses.

Fact 6.7 (7/8-approximation for Max-3SAT). *There is a simple $7/8$-approximation algorithm for* Max-3SAT.

Proof from the notes. Pick a uniformly random assignment. Fix a clause C with three literals. The only way C is unsatisfied is if all three literals evaluate to false, which happens with probability $1/8$. Hence $\Pr[C \text{ is satisfied}] = 7/8$.

By linearity of expectation, if m is the number of clauses then $\mathbb{E}[\#\text{ satisfied clauses}] = \frac{7}{8}m$. Since $\text{OPT}(\varphi) \le m$, this implies $\mathbb{E}[\#\text{ satisfied}] \ge \frac{7}{8}\,\text{OPT}(\varphi)$. Therefore some assignment achieves at least $\frac{7}{8}\,\text{OPT}(\varphi)$. (The notes remark that this randomized method can be derandomized using standard techniques such as conditional expectation.) ∎

Definition 6.8 (Max-Cut). Given an undirected graph $G = (V, E)$, the objective is to find a cut $S \subseteq V$ maximizing the number of edges crossing the cut, i.e. $|E(S, V \setminus S)|$.

Fact 6.9 (1/2-approximation for Max-Cut). *The random partition (put each vertex in S independently with probability $1/2$) achieves expected cut value $|E|/2$, hence gives a $1/2$-approximation.*

Definition 6.10 (Max-3XOR). An instance is a system of XOR constraints of the form $x_i \oplus x_j \oplus x_k = b$ with variables in $\{0, 1\}$. The objective is to satisfy as many equations as possible.

Fact 6.11 (1/2-approximation for Max-3XOR). *A uniformly random assignment satisfies each XOR equation with probability $1/2$, hence achieves expected value $m/2$.*

These simple algorithms prompt the natural question in the notes: *can we do better?* For several problems, the PCP theorem implies that the above constants are essentially the best possible unless $\mathsf{P} = \mathsf{NP}$.

THEOREM 6.12 (Håstad (informal statement from the notes)). *For every constant $\varepsilon > 0$:*

- *It is NP-hard to distinguish between a satisfiable 3CNF formula and one in which at most a $\frac{7}{8} + \varepsilon$ fraction of clauses can be satisfied.*
- *It is NP-hard to distinguish between a* MAX-3XOR *instance in which at least a $(1-\varepsilon)$ fraction of equations can be satisfied and one in which at most a $(\frac{1}{2} + \varepsilon)$ fraction can be satisfied.*

6.5 Max-kCSP and the PCP–approximation equivalence

Many approximation problems can be phrased uniformly as constraint satisfaction.

DEFINITION 6.13 (MAX-kCSP). Fix a predicate $\Phi : \{0,1\}^k \to \{0,1\}$. An instance consists of m constraints, where the t-th constraint applies Φ to some k-tuple of literals. The goal is to maximize the number of satisfied constraints.

The notes list MAX-3SAT, MAX-3XOR, and MAX-CUT as special cases.

THEOREM 6.14 (PCPs $\Rightarrow$ hardness of approximation (idea)). *Suppose $L \in \mathrm{PCP}(R, q, \Sigma, c, s)$ and that L is NP-hard (e.g. $L = \mathrm{SAT}$). Then there is a polynomial-time reduction from L to a gap version of* MAX-qCSP *that distinguishes instances with $\mathrm{OPT} \geq c\,m$ from those with $\mathrm{OPT} \leq s\,m$.*

Proof sketch from the notes. Fix a PCP verifier V for L. For each random string $r \in \{0,1\}^R$, the verifier deterministically chooses query positions $i_1(r), \dots, i_q(r)$ and applies a decision predicate $\varphi_r : \Sigma^q \to \{0,1\}$ to the q answers.

Now build a MAX-qCSP instance as follows:

- There is a variable for each proof location.
- For each $r \in \{0,1\}^R$, add a constraint that reads the q variables $\pi_{i_1(r)}, \dots, \pi_{i_q(r)}$ and is satisfied iff $\varphi_r(\pi_{i_1(r)}, \dots, \pi_{i_q(r)}) = 1$.

Let $m = 2^R$ be the number of constraints. For any assignment π to the variables, the fraction of satisfied constraints is exactly the verifier's acceptance probability $\Pr_r[V^\pi(x) \text{ accepts}]$.

If $x \in L$, completeness gives an assignment satisfying at least a c-fraction of constraints. If $x \notin L$, soundness says every assignment satisfies at most an s-fraction. Thus distinguishing $x \in L$ from $x \notin L$ reduces to the stated gap approximation problem. $\square$

THEOREM 6.15 (hardness of approximation $\Rightarrow$ PCPs (idea)). *Conversely, suppose there exists a predicate family for which it is NP-hard to distinguish* MAX-kCSP *instances with $\mathrm{OPT} \geq c\,m$ from those with $\mathrm{OPT} \leq s\,m$ for constants $c > s$. Then $\mathsf{NP} \subseteq \mathrm{PCP}(\log m, k, \Sigma, c, s)$.*

Proof sketch from the notes. Let f be the polynomial-time reduction from SAT to the above gap MAX-kCSP problem. On input a formula ϕ, the PCP verifier computes the instance

$I = f(\phi)$ with m constraints. It then chooses a uniformly random constraint index $t \in [m]$ (using $\log m$ random bits), queries the k proof positions referenced by that constraint, and accepts iff the constraint is satisfied.

If ϕ is satisfiable, there exists a proof/assignment that satisfies at least $c\,m$ constraints, so the verifier accepts with probability at least c. If ϕ is unsatisfiable, every assignment satisfies at most $s\,m$ constraints, so the verifier accepts with probability at most s. This is exactly a PCP with k queries and $\log m$ random bits. ∎

Taken together, Theorems 6.14 and 6.15 explain why PCPs and approximation hardness are essentially two views of the same phenomenon.

6.6 The baby PCP: an exponential-size PCP for quadratic equations

The remainder of the notes sketch a weaker PCP result that already shows the coding ideas behind the PCP theorem.

Theorem 6.16 (baby PCP (as stated in the notes)). $\mathsf{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), O(1))$.

Because $R(n) = \mathrm{poly}(n)$, the generic bound $q \cdot 2^R$ allows the proof length to be exponential in n; hence the name "exponential-size PCP" in the notes.

Theorem 6.17 (QuadEq is NP-complete). *Let* QuadEq *be the following decision problem. An instance is a system of quadratic equations over $\mathbb{F}_2$ in variables $x_1, \dots, x_n$, where each equation has total degree at most 2. Decide whether the system has a satisfying assignment. Then* QuadEq *is NP-complete.*

Proof sketch from the notes. Membership in NP is immediate: a candidate assignment can be checked in polynomial time. For hardness, start from a 3CNF formula and first convert it into an equivalent formula of fan-in 2 by replacing each 3-input gate by a small binary tree of 2-input gates. Introduce a new variable for the output of each internal gate and add constraints that enforce the correct relationship between a gate's inputs and output.

Over $\{0, 1\}$, the standard arithmetizations used in the notes are

$$\text{AND:} \quad y = x_1x_2, \qquad \text{OR:} \quad y = 1 - (1 - x_1)(1 - x_2).$$

Both constraints are quadratic. Finally, require that the output variable for the root gate equals 1. The resulting quadratic system is satisfiable iff the original Boolean formula is satisfiable. ∎

We now outline an oracle proof π and a constant-query verifier for QuadEq. The high-level idea is:

1. encode a satisfying assignment x in a way that lets us query *linear combinations* of its bits;
2. also encode all quadratic products x_ix_j;
3. test (locally) that these encodings are valid and consistent;
4. test (globally) that the encoded assignment satisfies the quadratic system.

6.7 Hadamard code and local decoding

Definition 6.18 (Hadamard code). For $u \in \{0,1\}^n$, define the Hadamard encoding $\mathrm{Enc}(u) \in \{0,1\}^{2^n}$ as the truth table of the function $H_u : \{0,1\}^n \to \{0,1\}$ given by

$$H_u(y) = \langle u, y \rangle \overset{\text{def}}{=} \sum_{i=1}^{n} u_i y_i \pmod 2.$$

We view $\mathrm{Enc}(u)$ as an array indexed by $y \in \{0,1\}^n$, so $\mathrm{Enc}(u)[y] = H_u(y)$.

Fact 6.19 (linearity). *Hadamard codewords are linear functions: for all $y, z \in \{0,1\}^n$, $H_u(y) \oplus H_u(z) = H_u(y \oplus z)$.*

Fact 6.20 (local decodability of Hadamard code). *Let $f : \{0,1\}^n \to \{0,1\}$ be a received word such that $\Pr_y[f(y) \neq H_u(y)] \leq \varepsilon$ for some u. Then for any target index $a \in \{0,1\}^n$, there is a 2-query randomized procedure that outputs $H_u(a)$ with probability at least $1 - 2\varepsilon$.*

Proof from the notes. To decode $H_u(a)$, pick a uniformly random $y \in \{0,1\}^n$ and query $f(y)$ and $f(y \oplus a)$. Output $f(y) \oplus f(y \oplus a)$.

If both queried values are correct, then

$$f(y) \oplus f(y \oplus a) = H_u(y) \oplus H_u(y \oplus a) = H_u(a),$$

using linearity. By a union bound, the probability that at least one of the two queries is corrupted is at most 2ε. □

6.8 The proof for QuadEq: encoding linear and quadratic terms

Let $x \in \{0,1\}^n$ be a satisfying assignment for a QuadEq instance. The proof π is split into two parts.

- $\pi_1 = \mathrm{Enc}(x)$, the Hadamard encoding of x. Intuitively this contains *all linear combinations* of the bits of x.
- $\pi_2 = \mathrm{Enc}(z)$ where $z = x \otimes x \in \{0,1\}^{n^2}$ is the length-n^2 string consisting of all quadratic products $z_{ij} = x_i x_j$. This part contains *all linear combinations* of the quadratic monomials.

Here $x \otimes x$ is viewed as an $n \times n$ matrix flattened into a length n^2 vector. For $s, r \in \{0,1\}^n$, we write $s \otimes r \in \{0,1\}^{n^2}$ for the outer product with (i,j)-entry $(s \otimes r)_{ij} = s_i r_j$.

6.9 The verifier: three tests

The notes describe three conceptual tests. The complete verifier would interleave them and amplify as needed; here we present each in isolation.

1. **Linearity test:** check that π_1 and π_2 are (close to) Hadamard codewords.

2. **Tensor test:** check that the decoded strings satisfy $\mathrm{Dec}(\pi_2) = \mathrm{Dec}(\pi_1) \otimes \mathrm{Dec}(\pi_1)$.
3. **Feasibility test:** check that $\mathrm{Dec}(\pi_1)$ satisfies the quadratic system.

The key constraint is that everything must be checkable with $O(1)$ queries.

6.10 Feasibility test via a random linear combination

Assume for the moment that the linearity and tensor tests have succeeded, so that π_1 and π_2 behave like encodings of some x and $z = x \otimes x$. We now want to test that (x, z) satisfies all quadratic equations.

Each quadratic equation over $\mathbb{F}_2$ can be written as

$$\langle A, z\rangle \oplus \langle b, x\rangle = c, \tag{6.1}$$

where $A \in \{0,1\}^{n^2}$ selects which quadratic monomials appear, $b \in \{0,1\}^n$ selects which linear terms appear, and $c \in \{0,1\}$ is the constant term. Because π_1 and π_2 are Hadamard encodings, we can evaluate the left hand side of (6.1) with two oracle queries: query $\pi_2[A]$ to obtain $\langle A, z\rangle$ and query $\pi_1[b]$ to obtain $\langle b, x\rangle$, then check whether their XOR equals c.

However, there may be many equations, so the notes propose a *global* test: take a uniformly random subset of equations and check their XOR.

Formally, if the system has equations $\langle A^{(t)}, z\rangle \oplus \langle b^{(t)}, x\rangle = c^{(t)}$ for $t = 1, \ldots, m$, pick $\sigma \in \{0,1\}^m$ uniformly at random and define

$$A^* = \bigoplus_{t=1}^{m} \sigma_t A^{(t)}, \qquad b^* = \bigoplus_{t=1}^{m} \sigma_t b^{(t)}, \qquad c^* = \bigoplus_{t=1}^{m} \sigma_t c^{(t)}.$$

The verifier checks whether $\langle A^*, z\rangle \oplus \langle b^*, x\rangle = c^*$ using two oracle queries.

- **Completeness:** if all equations hold, then the XOR of any subset holds, so the test always accepts.
- **Soundness:** if at least one equation is violated by (x, z), then the XOR of a uniformly random subset is violated with probability exactly $1/2$ (equivalently, an odd number of violated equations is included with probability $1/2$).

6.11 Tensor test

The tensor test checks that π_2 encodes the pointwise products of the decoded π_1.

Assume $\pi_1 = \mathrm{Enc}(x)$ and $\pi_2 = \mathrm{Enc}(z)$. If $z = x \otimes x$, then for all $s, r \in \{0,1\}^n$,

$$\begin{aligned}
\pi_2[s \otimes r] &= \langle s \otimes r, x \otimes x\rangle && (6.2)\\
&= \sum_{i,j} (s_i r_j)(x_i x_j) && (6.3)\\
&= \Big(\sum_i s_i x_i\Big)\Big(\sum_j r_j x_j\Big) \;=\; \pi_1[s] \cdot \pi_1[r] \qquad (\text{over } \mathbb{F}_2). && (6.4)
\end{aligned}$$

This suggests the following 3-query test.

Definition 6.21 (tensor test). Pick uniformly random $s, r \in \{0,1\}^n$. Query $\pi_1[s]$, $\pi_1[r]$, and $\pi_2[s \otimes r]$. Accept iff $\pi_2[s \otimes r] = \pi_1[s] \cdot \pi_1[r]$.

The notes state that this test has completeness 1 and soundness at most 3/4 (under the assumption that the tables are valid Hadamard encodings). The key calculation is that if $z \neq x \otimes x$, then with noticeable probability over s, r, the inner products $\langle s \otimes r, z \rangle$ and $\langle s \otimes r, x \otimes x \rangle$ differ; the notes lower bound this disagreement probability by 1/4, giving a 3/4 upper bound on the acceptance probability.

6.12 Linearity test: BLR

To justify treating π_1 and π_2 as (noisy) Hadamard codewords, we need a local test for linearity. The notes invoke the classic BLR test.

Definition 6.22 (BLR linearity test). Given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$: pick uniformly random $x, y \in \{0,1\}^n$, query $f(x)$, $f(y)$, and $f(x \oplus y)$, and accept iff $f(x) \oplus f(y) = f(x \oplus y)$.

Theorem 6.23 (BLR (informal statement from the notes)). *If f is ε-far (in relative Hamming distance) from every linear function, then the BLR test rejects with probability $\Theta(\varepsilon)$.*

In particular, passing the BLR test with high probability implies that f is close to some Hadamard codeword H_u, and then Fact 6.20 allows (local) decoding of linear combinations of the underlying message u.