A Mini-Course on Probabilistically Checkable Proofs

Orr Paradise[*]

The 2018 Amos de-Shalit Summer School
Weizmann Institute of Science

Contents

[*]http://www.wisdom.weizmann.ac.il/~orrp/

Part I

Sunday, September 2nd 2018

This course does not directly follow any specific textbook (and therefore all errors and omissions are the teacher's). Two great resources for additional reading (and studying complexity theory in general) are [Gol08, AB09].

1 Prelude: Computation and Verification

What can be computed efficiently? This question stands at the heart of the field of computational complexity. Complexity theorists aim to answer questions of this type, a famous example being "If a solution for a problem can be verified efficiently, can it also be found efficiently?".

Definition 1.1 ($\mathcal{NP}$)**.** A language L is *efficiently verifiable* if there exists a Turing Machine V with the following properties:

- *Completeness*: For all $x \in L$ there exists a *witness* w of length polynomial in $|x|$ such that $V(x, w) = 1$.
- *Soundness*: For all $x \notin L$ and any w it holds that $V(x, w) = 0$.
- *Efficiency*: V runs in polynomial time (both in x and in w, since their lengths are polynomially related).

We denote by $\mathcal{NP}$ the *class of efficiently verifiable decision problems.*

This the go-to definition of a proof, in the computational sense. While there are many ingenuous theories and tantalizing open questions that build upon this basic object/proof-system, we argue that there are other notions of proof-systems that are worth considering.

1.1 Randomness in computation (and verification)

In a previous course in algorithms you may have encountered the power of randomness in computation. For example, randomized quicksort (choosing a pivot randomly) has significantly better worst-case complexity than its deterministic version ($O(n\log n)$ vs. $O(n^2)$). Complexity theorists have not yet settled the matter of how randomness affects the complexity of *solving* decision problems[1]. But what about *verifying* them? This is exactly the idea underlying Probabilistically Checkable Proofs.

1.2 Decision problems, a (re)formality

Formally, a decision problem is a language $L \subseteq \{0,1\}^*$, the "problem" being to decide whether a certain input string $x \in \{0,1\}^*$ is a word in the language L (i.e. $x \in L$) or not. We will use a different formulation, which is arguably more natural in the modern context.

Definition 1.2. A *decision (promise) problem* is a pair $\Pi = (\Pi_{\texttt{YES}}, \Pi_{\texttt{NO}})$ where $\Pi_{\texttt{YES}}, \Pi_{\texttt{NO}} \subseteq \{0,1\}^*$ are disjoint. $\Pi_{\texttt{YES}}$ is referred to as the *set of YES-cases* and $\Pi_{\texttt{NO}}$ is the *set of NO-cases.* The *promise* is the set $\Pi_{\texttt{YES}} \sqcup \Pi_{\texttt{NO}}$, and the goal is to decide whether an input string from the promise is a YES-case or a NO-case. That is: "Given $x \in \Pi_{\texttt{YES}} \sqcup \Pi_{\texttt{NO}}$, is $x \in \Pi_{\texttt{YES}}$?"

Sometimes, the promise simply allows us to ignore inputs that aren't well-formed when designing our algorithm: for example, when designing an algorithm for undirected graph Hamiltonicity, we can add the promise that the input adjacency matrix is symmetric. This difference is indeed a syntactic one, and the problem's complexity remains the same. But other types of promises can make a big difference, as we will see later.

[1]The question is "If a problem can be solved by a probabilistic Turing Machine (with high probability), can it also be solved efficiently by a deterministic Turing Machine?

2 Probabilistically Checkable Proofs

Without further ado, we introduce the lead actor in our play in four acts.

Definition 2.1 ($\mathcal{PCP}$)**.** A decision problem $\Pi = (\Pi_{\texttt{YES}}, \Pi_{\texttt{NO}})$ has a *probabilistically checkable proof (PCP) system using randomness complexity* $r(n)$ *and query complexity* $q(n)$ if there exists a probabilistic Turing Machine *verifier* V with the following properties:

- *Completeness*: For all $x \in \Pi_{\texttt{YES}}$ there exists a *proof* π such that $V^{\pi}(x) = 1$.
- *Soundness*: For all $x \in \Pi_{\texttt{NO}}$ and any π, $V^{\pi}(x) = 0$ with high probability (with probability at least 1/2).
- *Efficiency*: V runs in time polynomial in $|x|$, tosses $r(|x|)$ random coins and queries π in $q(|x|)$ locations.

We denote by $\mathcal{PCP}(r, q)$ the class of decision problems that have a PCP system as above. For classes of functions R and Q, let $\mathcal{PCP}(R, Q) := \bigcup_{r\in R, q\in Q} \mathcal{PCP}(R, Q)$.

A "better" PCP is one in which r and q are small: the idea is to verify that the entire proof is correct while only reading a few bits from it. This hints that the proof should be encoded in some special way, so that a random local view of the proof tells us something about its correctness (a global property). We tell a small tale that might help with understanding this paradigm, and warn that this is far from how it is actually used (in complexity theory):

Tale 2.1*.* Alice wishes to compute the answer to a very difficult question, one that is infeasible to compute on her own personal computer. Amazon offers to run the computation for her on their powerful computers (for a small fee). But how can she be sure that they actually ran the computation and didn't simply say "True"? She could ask for the entire transcript of the computation, but in order to verify that they did not cheat in any step she must verify it in its entirety, which is at least as difficult as running the computation herself! Instead, she asks Amazon for a special encoding of the proof (a *probabilistically checkable proof*) that proves that their answer is correct. Sure, computing the proof might be difficult, but they have very strong computers, and once they send her the proof she can run the *verifier* (which is far more efficient than the computation itself) to verify the computation with a high degree of certainty (as high as she wishes, see Exercise 2.1).

Exercise 2.1 (Soundness amplification)*.* Prove that if L has a PCP system with verifier V, then for all $t \in \mathbb{N}$, the rejection probability in the *soundness* condition can be amplified to to $1 - 2^{-t}$ at the cost of a t-factor increase in randomness and query complexities. *Guidline:* Consider the machine V^t that emulates t runs of V (with independent coins for each run) and accepts if and only if all emulated runs accepted.

2.1 An example

It might be helpful to see a 'toy' PCP in order to internalize its definition.

Definition 2.2 (`gap3SAT`)**.** `gap3SAT` is the problem of deciding 3CNF formula satisfiability under the promise that the input formula is either satisfiable, or that no assignment satisfies more than half of its clauses. That is, ϕ is a `NO`-case of `gap3SAT` if no assignment satisfies more than half of its clauses simultaneously, and is a `YES`-case if it satisfiable.

A peculiar problem, but interesting nonetheless (see 2.3). We construct a PCP system for it.

Algorithm 2.1. *Given access to an input formula* ϕ *and proof oracle* π*, the PCP verifier for* `gap3SAT` *samples a uniformly random clause* C *of* ϕ*, then reads the assignments for the three variables that appear in* C *and accepts if and only the assignment satisfies* C*.*

Exercise 2.2*.* Prove that the above algorithm is a PCP verifier for `gap3SAT`.

While the above example is indeed a PCP system, it does not encapsulate the main idea underlying useful PCPs, which is determining a global property (correctness of a proof) while making a few local observations (queries to the proof).

2.2 The power of PCPs

So we have a new and interesting proof system, and immediately how powerful it is: can it do something that NP proof systems can't do, or vice-versa? When reasoning about PCPs we must always keep in mind the resources available to them. The following observations are somewhat immediate:

Exercise 2.3. Prove the following:

- $\mathcal{PCP}(O(1), \texttt{poly}) = \mathcal{NP}$ (Exercise)
- $\mathcal{PCP}(\texttt{log}, O(1)) \subseteq \mathcal{NP}$ (Exercise. Guideline: the $\mathcal{NP}$-witness should be all possible answers given to the verifier when provided with a correct proof).

In this course we will prove $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$ by constructing a clever PCP system for an $\mathcal{NP}$-complete language. Astoundingly, NP proof systems are encompassed by even weaker PCP systems, as the remarkable result $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$ (which bears the well-earned name of *the PCP theorem*) is also known to be true. The latter is much more difficult to obtain: it was the culmination of a series of cutting-edge works that involved the construction of other cunning PCP systems, and some other remarkable ideas. This "settles" the inquiry into the power of PCP systems (compared to $\mathcal{NP}$ proof systems)

Theorem 2.1 (The PCP theorem)**.**

$$\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$$

However, PCPs continue to yield many useful results in other fields (such as cryptography and distributed algorithms) and their nature is still studied extensively.

2.3 A different view on the PCP theorem (if time permits)

In this course we will be study PCPs (and their power) as proof systems. In this section we show that PCPs themselves can be viewed as "combinatorial" decision problems, in a way. In fact, this perspective has yielded an alternative (and arguably simpler) proof of the PCP theorem, years after its original proof!

Definition 2.3 ($\varepsilon-\texttt{gap}q\texttt{CSP}$)**.** The *q-ary constraint satisfaction problem*: Given *constraints* $F = \{f : \{0,1\}^q \to \{0,1\}\}$, is there $x \in \{0,1\}^q$ that satisfies the constraints, i.e. that $f(x) = 1$ for all $f \in F$?

The *q-ary constraint satisfaction problem with gap ε ($\varepsilon-\texttt{gap}q\texttt{CSP}$)* is the above problem under the promise that the input constraints are either either satisfiable, or that no assignment satisfies more than an ε-fraction of them.

Claim 2.1. *The PCP theorem holds if and only if $\varepsilon-\texttt{gap}q\texttt{CSP}$ is $\mathcal{NP}$-hard.*

Proof. Assume that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q)$. Then there exists a PCP verifier V for SAT. We prove that $0.5-\texttt{gap}q\texttt{CSP}$ for some q. A formula ϕ of length n is reduced to an instance of $0.5-\texttt{gap}q\texttt{CSP}$ as follows. For any $R \in \{0,1\}^{O(\log(n))}$ we define a constraint $V_R : \{0,1\}^q : \{0,1\}$ by letting $V_R(a_1 \cdots a_q) = 1$ iff V accepts ϕ upon sampling coins R and receiving $a_1, \ldots, a_q$ as answers to its queries. Completeness and soundness of the reduction follow from completeness and soundness of the PCP. The reduction is efficient because computing each constraint requires emulating V on $2^q = O(1)$ inputs (and each emulation takes $\text{poly}(n)$ time), and there are a total of $2^{O(\log(n))} = \text{poly}(n)$ constraints to compute.

Now assume that $\varepsilon-\texttt{gap}q\texttt{CSP}$ is $\mathcal{NP}$-hard. We construct a PCP system for SAT as follows. Fix a formula ϕ (an instance of SAT), and let F be the result of its reduction to an instance of $\varepsilon-\texttt{gap}q\texttt{CSP}$. The PCP verifier V expects a satisfying assignment $\pi \in \{0,1\}^q$ to F as proof, and will simply uniformly sample a constraint in F and check that it's satisfied by π. Completeness is immediate, and to get soundness $1/2$ (instead of ε) we can use 2.1. Efficiency of V follows from the efficiency of the reduction, and notice that V makes q queries to the proof and uses $\log(|F|) = O(\log(n))$ randomness. ☐

Exercise 2.4. Prove that (0.5-)gap3SAT is $\mathcal{NP}$-hard if and only if $\varepsilon-\texttt{gap}q\texttt{CSP}$ is $\mathcal{NP}$-hard for some $\varepsilon > 0$.

Part II

Monday, September 3rd 2018

The task of a PCP is to allow verification of an entire proof by reading only a few random bits of it. In other words, we must decide a global problem while having access only to a small local view. This requires some special encoding of witnesses. Underlying a construction of such a code is a beautiful mathematical idea.

Boolean functions are all around us, if we look close enough:

- A formula is a Boolean function mapping assignments to truth values.
- A property of a graph (such as bipartiteness or Hamiltonicity) is a Boolean function that outputs 1 iff the graph (as an adjacency matrix) has the property.
- Voting rules are Boolean functions: democracy is the majority function, dictatorship is the indicator function.

In this lesson we will learn a new way to look at Boolean functions, showing us a powerful way to reason about computational problems. [O'D14] is an excellent place to look for further reading.

3 The geometry of Boolean functions

Remark 3.1. Usually we think of bits as taking values in $\{0,1\}$: 0 means "False" and 1 means "True". Equivalently (or isomorphically), from this point onwards we decide that bits take values in $\{\pm 1\}$, with 1 meaning "False" and -1 meaning "True". This unconventionality will make our notation much prettier later, though mathematically all will be equivalent.

The space of real-valued functions on n bits is $\mathbb{R}^{\{\pm1\}^n} \cong \mathbb{R}^{2^n}$. The standard inner product of two functions f, g as vectors in that space is given by $\langle f, g\rangle = \sum_{x\in\mathbb{Z}_2^n} f(x)\, g(x)$. Normalizing, the inner product of two functions is the expectation of their product on a uniformly random input:

$$\langle f, g\rangle := 2^{-n} \sum_{x\in\{\pm1\}^n} f(x)\, g(x) = \mathop{\mathbb{E}}_{x\leftarrow\{\pm1\}^n} [f(x)\, g(x)] \tag{1}$$

3.1 An orthonormal basis of multiplicative functions

With an inner product in place, we move on to the natural task of finding an orthonormal basis. An immediate observation is that since the inner product is a scaling of the standard one, then an appropriate scaling of the standard basis[2] forms an orthonormal basis.

We construct another, more useful orthonormal basis. For any $I \subseteq [n]$, the *Ith parity function* $\chi_I : \{\pm1\}^n \to \{\pm1\}$ is $\chi_I(x) := \prod_{i\in I} x_i$.[3]

Claim 3.1. *The 2^n parity functions form an orthonormal basis to the vector space of Boolean functions.*

Proof. Since the space has dimension 2^n, it suffices to show orthonormality. For all I, J,

$$\langle \chi_I, \chi_J\rangle = \mathbb{E}[\chi_I(x)\, \chi_J(x)] = \mathbb{E}\left[\prod_{i\in I} x_i \prod_{j\in J} x_j\right] = \mathbb{E}\left[\prod_{k\in I\triangle J} x_k\right] = \begin{cases} 1 & \text{if } I = J \\ 0 & \text{if } I \neq J \end{cases} \tag{2}$$

We used the fact that $x_k \cdot x_k = 1$ for the third equality, and that parity of a uniformly selected (nonempty) sequence of bits is equally likely to be "True" or "False". $\square$

[2]Recall that the standard basis of $\{\pm1\}^{\{\pm1\}^n}$ is the family of indicator functions $\{1_x\}_{x\in\{\pm1\}^n}$ where $1_x(y) = -1$ iff $x = y$

[3]The product of zero elements is defined to be 1.

As with any orthonormal basis, any function can be uniquely decomposed to linear combination of its elements $f = \sum_{I \subseteq [n]} \langle f, \chi_I \rangle \chi_I$. This decomposition is known as the *Fourier decomposition*, and the coefficient $\langle f, \chi_S \rangle =: \widehat{f}(S)$ is called *f's Ith Fourier coefficient*. Sometimes we can say things about f just by looking at its Fourier coefficients.

3.2 Some examples for the usefulness of the Fourier decomposition

It is immediate that the *mean* of f is its $\emptyset$th ("first") coefficient: $\mathbb{E}[f(x)] = \langle f, \chi_\emptyset \rangle = \widehat{f}(\emptyset)$.

Claim 3.2 (Parseval's equality)**.**

$$\langle f, f \rangle = \sum_I \widehat{f}(I)$$

Proof.

$$\langle f, f \rangle = \left\langle \sum_I \widehat{f}(I) \chi_I, \sum_J \widehat{f}(J) \chi_J \right\rangle = \sum_{I,J} \widehat{f}(I) \widehat{f}(J) \langle \chi_I, \chi_J \rangle = \sum_I \widehat{f}(I)^2$$

□

Corollary 3.1. *Using Parseval's equality,*

1. $\mathrm{Var}[f] \coloneqq \mathbb{E}\left[f^2(x)\right] - \mathbb{E}[f(x)]^2 = \sum_{I \subseteq [n], I \neq \emptyset} \widehat{f}(I)^2$

2. *For any Boolean f, $\sum_I \widehat{f}(I) = \mathbb{E}\left[f(x)^2\right] = 1$*

Exercise 3.1 (Plancheral's equality and the covariance). Prove $\langle f, g \rangle = \sum_I \widehat{f}(I) \widehat{g}(I)$, and use that to express the covariance of f and g in terms of their Fourier coefficients.

4 Multiplicative Boolean functions

So far we've only viewed Boolean functions as glorified 2^n-dimensional real vectors (and still, we've learned quite a lot). Recall that the domain of these functions is the vector space $\{\pm 1\}^n$, with pointwise multiplication corresponding to vector addition. Boolean functions are then functions from $\{\pm 1\}^n$ to its underlying field ($\{\pm 1\}$). We're particularly interested in Boolean functions that happen to be linear forms, which in our case means that they are multiplicative[4]. The parity functions (the orthonormal basis extensively used above) were all multiplicative, and it turns out that they are the *only multiplicative* Boolean functions.

Claim 4.1. *$f : \{\pm 1\}^n \to \{\pm 1\}$ is multiplicative if and only if there exists $I \subseteq [n]$ such that $f = \chi_I$.*

Proof. Recall that we are working above the field $(\{\pm 1\}, \times, \wedge)$.[5] The standard inner product above $\{\pm 1\}^n$ is $x \odot y \coloneqq \prod_{i=1}^n x_i \wedge y_i$ (this proof is the only time we will use this inner product). Riesz's representation theorem states that $f : \{\pm 1\}^n \to \{\pm 1\}$ is multiplicative if and only if there exists $u \in \{\pm 1\}^n$ such that $f(x) = x \odot u$ for all x. Notice that for all $u \in \{\pm 1\}^n$,

$$x \odot u = \prod_{i=1}^n x_i \wedge u_i = \prod_{i : u_i = -1} x_i$$

Mapping each $u \in \{\pm 1\}^n$ to χ_I for $I = \{i \mid u_i = -1\} \subseteq [n]$, we have a bijection between the space of multiplicative Boolean functions and the set of parity functions.

□

[4] f is multiplicative if $f(xy) = f(x) f(y)$, where xy denotes the pointwise multiplication of vectors x and y

[5] $\wedge$ denotes the logical AND operation: $a \wedge b = -1$ iff $a = b = -1$, for all $a, b \in \{\pm 1\}$

Simpler proof. Let M_n denote the space of multiplicative Boolean functions. M_n is dual to $\{\pm 1\}^n$ which is of dimension n, hence the dimension of M_n is n. An n-dimensional space over a field of size 2 has 2^n elements, hence the 2^n parity functions are the only elements in M_n. $\square$

So we were well acquainted with the multiplicative functions all along! They will accompany us for the rest of our journey.

4.1 The Hadamard code

We're trying to construct an encoding of witnesses in a way that will allow us to verify their correctness by reading only a few bits from their encoding. A prerequisite to this encoding is that different witnesses should have very different encodings. Otherwise, a single difference in the witness (for example the assignment to a certain variable) would likely go undetected if we only read a few random bits!

If we had a set of words that are very different, we could assign each witness a different word, satisfying the necessary property above. Let's formalize things a bit.

Definition 4.1 (Hamming distance)**.** The *(relative) Hamming distance* of two length m strings $u, v \in \{\pm 1\}^m$ is the number of locations in which they differ, normalized by their length:

$$\delta(u, v) \coloneqq \underset{i \leftarrow [m]}{\mathbb{P}} [u_i \neq v_i]$$

.

The distance of a function f to a set of functions S is the minimal distance to any element of the set

$$\delta(f, S) \coloneqq \min_{g \in S \cap \text{Domain}(f)} \delta(f, g)$$

The distance from the empty set is always 1.

If $\delta(f, g) \geq \varepsilon$ then we say that f is *ε-far* from g. If $\delta(f, g) \leq \varepsilon$ then f is *ε-close* to g.

Claim 4.2. *For any $f, g : \{\pm 1\}^n \to \{\pm 1\}$, $\langle f, g \rangle = 1 - 2\delta(f, g)$*

Proof.

$$\mathbb{E}[f(x) g(x)] = \mathbb{P}[f(x) = g(x)] - \mathbb{P}[f(x) \neq g(x)] = 1 - 2\delta(f, g)$$

$\square$

Corollary 4.1 (Distance of the Hadamard code)**.** *The distance between any two multiplicative Boolean functions is $1/2$, as they are orthonormal.*

The multiplicative functions are far apart, so the encoding $u \mapsto \langle \cdot, u \rangle$, also known as the Hadamard encoding,[6] might work for us. Note that it is quite costly: words of lenght n are encoded by truth tables which have length 2^n, and exponential blowup! There are codes that have constant minimum distance between codewords, and a much more reasonable (even linear) blowup in codeword length.

Large minimal distance is not the only thing we need in our encoding. If we wish for proofs to be codewords, we also need a way to efficiently verify that the proof is a valid codeword (i.e. a multiplicative function), and an efficient way to decode the proof back to the original witness. Since we are dealing with PCPs, "efficient" always means "make local probes and be correct with high probability".

[6]Named after Jacques Hadamard, this code was used by NASA to transmit photos from Mars back to Earth in 1971!

Part III

Tuesday, September 4th 2018

Our task for today is constructing efficient procedures for testing and decoding the Hadamard code.

5 A connection: property testing

The field of property testing is concerned with the superfast algorithms for approximate decision making. In other words, algorithms that are highly efficient (in access to the input, and usually runtime as well) that (mostly) decide problems correctly with high probability. Why "mostly"? Property testers usually have extremely restricted access to the input (for example, they may read only 4 bits) and so we ask of them to decide whether an input has the property or is *far* (in Hamming distance) from having it.

Definition 5.1 (Tester)**.** A *property* $\Pi = \bigcup_{n\in\mathbb{N}} \Pi_n$ is a set of Boolean functions where $\Pi_n = \Pi \cap \{f : \{\pm 1\}^n \to \{\pm 1\}\}$. A *tester* for property Π (with one-sided error) a probabilistic oracle machine T that on input $n \in \mathbb{N}$ and $\varepsilon > 0$ and oracle access to $f : \{\pm 1\}^n \to \{\pm 1\}$ satisfies

- *Completeness:* If $f \in \Pi$ then T accepts with probability 1.
- *Soundness:* If f is ε-far from Π_n then T rejects with probability at least 1/2.

We're interested in the amount of locations in f read by T, and the number of random bits used by T. $q(n)$ (resp. $r(n)$) is the maximal amount of locations read (resp. random bits used) by T on inputs in $\{g : \{\pm 1\}^n \to \{\pm 1\}\}$.

Research in property testing goes both in the positive direction (the construction and analysis of testers) and the negative one (proving that certain problems have no efficient testers).

5.1 Property testing and PCPs

The above definition is reminiscent of that of PCPs, in that global decisions should be made correctly with high probability, based only on local views of the input. Indeed, PCPs and property testers are strongly entwined: property testers are often used by PCPs' verification procedure, and on the other hand PCPs can be seen as testers for the property of "being a correct proof" (but this is pushing it). In fact, in the construction of highly efficient PCPs (e.g., that of the PCP theorem) a creature that is a perfect blend between PCPs and property testers (known as a *PCP of Proximity*) is used: such creatures are required to verify whether an input is a `YES` case or is ε-far from being one.

6 Testing and decoding the Hadamard code

Remark 6.1. It's worth noting that there are alternative proofs for the Hadamard code local tester [Gol18, Section 2] that are not Fourier-analytic and result in a more general result.

6.1 Testing the Hadamard code

Algorithm 6.1 (The Hadamard code (local) tester)**.** *Given oracle access to a function $f : \{\pm 1\}^n \to \{\pm 1\}$, the tester*

1. *Samples $x, y \leftarrow \{\pm 1\}^n$ uniformly and independently.*
2. *Queries f for the values $f(x)$, $f(y)$, $f(xy)$.*
3. *Accepts if and only if $f(x) f(y) = f(xy)$*

Claim 6.1. *For all $f : \{\pm 1\}^n \to \{\pm 1\}$, the following holds:*

- Completeness: *If f is multiplicative, the tester accepts with probability* 1.
- Soundness: *If f is δ-far from multiplicative, the tester rejects with probability at least δ.*

Exercise 6.1. Use the above claim to derive a tester for the property $\mathrm{Mul} = \{f : \{\pm 1\}^n \to \{\pm 1\} \mid f \text{ is multiplicative}\}$ that makes constant queries to its input and uses poly$n$ randomness. Guideline: use soundness amplification from lecture 1.

Proof of 6.1. We prove soundness. Let $f : \{\pm 1\}^n \to \{\pm 1\}$. The first step is a simple form of *arithmetization*: we analyze a mathematical expression that is somehow related to the probability of rejecting f. Noticing that

$$f(x) f(y) f(xy) = \begin{cases} 1 & \text{if } f(x) f(y) = f(xy) \\ -1 & \text{otherwise} \end{cases}$$

We have

$$\begin{aligned}\mathbb{E}[f(x) f(y) f(xy)] &= \mathbb{P}[f(x) f(y) = f(xy)] - \mathbb{P}[f(x) f(y) \neq f(xy)] \\ &= \mathbb{P}[\text{The test accepts } f] - \mathbb{P}[\text{The test rejects } f] = 1 - 2\,\mathbb{P}[\text{The test rejects } f]\end{aligned}$$

So our goal is to show that $\mathbb{E}[f(x) f(y) f(xy)] < 1 - 2\delta$. We start by expressing that expectation in terms of f's Fourier coefficients:

$$\begin{aligned}
\mathbb{E}[f(x) f(y) f(xy)] &= = \mathop{\mathbb{E}}_{x,y}\left[\left(\sum_I \widehat{f}(I) \chi_I(x)\right)\left(\sum_J \widehat{f}(J) \chi_J(y)\right)\left(\sum_K \widehat{f}(K) \chi_K(xy)\right)\right] \\
&= \mathop{\mathbb{E}}_{x,y}\left[\sum_{I,J,K} \widehat{f}(I) \chi_I(x) \widehat{f}(J) \chi_J(y) \widehat{f}(K) \chi_K(xy)\right] \\
&= \sum_{I,J,K} \widehat{f}(I) \widehat{f}(J) \widehat{f}(K) \mathop{\mathbb{E}}_{x,y}[\chi_I(x) \chi_J(y) \chi_K(xy)] \\
\text{The parity functions are multiplicative } &= \sum_{I,J,K} \widehat{f}(I) \widehat{f}(J) \widehat{f}(K) \mathop{\mathbb{E}}_{x,y}[\chi_I(x) \chi_J(y) \chi_K(x) \chi_K(y)] \\
x, y \text{ are independent } &= \sum_{I,J,K} \widehat{f}(I) \widehat{f}(J) \widehat{f}(K) \mathop{\mathbb{E}}_x[\chi_I(x) \chi_K(x)] \mathop{\mathbb{E}}_y[\chi_J(y) \chi_K(y)] \\
&= \sum_{I,J,K} \widehat{f}(I) \widehat{f}(J) \widehat{f}(K) \langle \chi_I, \chi_K \rangle \langle \chi_J, \chi_K \rangle \\
\text{The parity functions are orthonormal } &= \sum_I \widehat{f}(I)^3
\end{aligned} \tag{3}$$

Let f^* be the multiplicative function closest to f. As we've seen, there must be $I^* \subseteq [n]$ such that $f^* = \chi_{I^*}$ is the I^*th parity function. Recall that

$$\widehat{f}(I) \coloneqq \langle f, \chi_I \rangle = 1 - 2\delta(f, \chi_I) \tag{4}$$

So f being closest to χ_{I^*} of all multiplicative functions means that $\widehat{f}(I^*)$ is the largest of f's Fourier coefficients. Thus, from (3)

$$|\mathbb{E}[f(x) f(y) f(xy)]| \leq \sum_I \left|\widehat{f}(I)^3\right| \leq \widehat{f}(I^*) \sum_I \widehat{f}(I)^2 = \widehat{f}(I^*) \cdot 1 = 1 - 2\delta(f, \chi_{I^*})$$

Where we used the fact that $\sum_I \widehat{f}(I)^2 = 1$ (seen in the previous lecture) and (4). $\square$

6.2 Decoding the Hadamard code

The ability to check that a given proof consists of some codeword is essential, but is not enough. We should also be able to decode the original message (e.g. assignment to a formula) from a given codeword. In fact, we need to do this "locally": since we need to read only a few bits from the message (reading it in its entirety is too costly for a PCP), we need to be able to decode a few bits in the message by looking at only a few bits in the codeword. This task (known as "local decoding") is quite simple in the Hadamard code: the message u is encoded $\langle \cdot, u\rangle$, so to read u_i we need to read $\langle e_i, u\rangle$.

But the key idea is to perform local decoding *in the presence of noise*. That is, given access to f a slightly (uniformly) noised version of the codeword $\widetilde{f}$, output $\widetilde{f}$'s value at a specific location x.

Algorithm 6.2 (The Hadamard code (local) decoder)**.** *Given oracle access to function $f : \{\pm 1\}^n \to \{\pm 1\}$ and location $x \in \{\pm 1\}^n$ as explicit input, sample $r \leftarrow \{\pm 1\}^n$ and output $f(xr)\, f(r)$*

Claim 6.2. *If f is δ-close to a multiplicative function $\widetilde{f}$ and any location $x \in \{\pm 1\}^n$, the Hadamard decoder errs (i.e. outputs a value other than $f(x)$) with probability at most 2δ.*

Proof. The key observation is that for fixed $x \in \{\pm 1\}^n$ and uniformly random $r \leftarrow \{\pm 1\}^n$, xr is distributed uniformly in $\{\pm 1\}^n$ (exercise). Then we note that

$$\widetilde{f}(xr)\, \widetilde{f}(r) = \widetilde{f}(xrr) = \widetilde{f}(x)$$

from multiplicativity of $\widetilde{f}$ (and since rr is the all-ones vector), and additionally if f and $\widetilde{f}$ agreed on both xr and r then the decoder outputs the correct value, and each event occurs with all but probability δ by definition of the Hamming distance. For those who prefer symbols to words:

$$\mathop{\mathbb{P}}_{r}\left[f(xr) \neq \widetilde{f}(xr)\right] = \mathop{\mathbb{P}}_{r}\left[f(r) \neq \widetilde{f}(r)\right] = \delta$$

Therefore

$$\begin{aligned}\mathop{\mathbb{P}}_{r}[\text{Error}] = \mathop{\mathbb{P}}_{r}\left[f(xr)\, f(r) \neq \widetilde{f}(x)\right] &= \mathop{\mathbb{P}}_{r}\left[f(xr)\, f(r) \neq \widetilde{f}(xr)\, \widetilde{f}(r)\right] \\ &\leq \mathop{\mathbb{P}}_{r}\left[f(xr) \neq \widetilde{f}(xr) \vee f(r) \neq \widetilde{f}(r)\right] \\ &\leq \mathop{\mathbb{P}}_{r}\left[f(xr) \neq \widetilde{f}(xr)\right] + \mathop{\mathbb{P}}_{r}\left[f(r) \neq \widetilde{f}(r)\right] \leq 2\delta\end{aligned}$$

□

Exercise 6.2 ("Test and decode" (also known as self-correction)).

Algorithm 6.3. *Fix an integer q. Given inputs $x_1, \ldots, x_q \in \{\pm 1\}^n$ and oracle access to function $f : \{\pm 1\}^n \to \{\pm 1\}$:*

1. *Perform $t(q) \geq 1/\log(4q/(4q-1))$ trials of Algorithm 6.1 using independent ("fresh") randomness. If any of these rejected, reject.*
2. *Uniformly sample $r \leftarrow \{\pm 1\}^n$. For all $i \in [q]$ output $f(x_i + r) - f(r)$.*

Suppose f is δ-close to a multiplicative function $\widetilde{f}$. Prove that with probability at least 1/2, Algorithm 6.3 either rejects or outputs $\widetilde{f}(x_1) \ldots \widetilde{f}(x_q)$. Guideline:

- If $\delta > 1/(4q)$ then with probability at least 1/2 (over the coins of the trials of Algorithm 6.1) some trial rejects. Hint: Upper bound the probability that all t independent trials accepted.
- If $\delta \leq 1/(4q)$ then with probability at least 1/2 (over the sampling of r), the algorithm outputs $\widetilde{f}(x_1) \ldots \widetilde{f}(x_q)$. Hint: Lower bound using the union bound.

Part IV
Wednesday, September 5th 2018

In this final meeting we will construct and analyze an interesting PCP system for an $\mathcal{NP}$-complete language that makes a constant number of queries and uses polynomial randomness, giving us $\mathcal{NP} \subseteq \mathcal{PCP}\left(O\left(1\right), \texttt{poly}\right)$.

7 Quadratic equations over the field of two elements

The go-to (and first known) $\mathcal{NP}$-complete language is `SAT`, whose `YES`-cases are satisfiable CNF formulae and `NO`-cases are unsatisfiable ones. We define a different $\mathcal{NP}$-complete language and construct a PCP system for it, and notice that the choice of language is immaterial for our result, as long as the language is $\mathcal{NP}$-hard:

Exercise 7.1. Prove that if there exists a PCP system of complexities q, r for some $\mathcal{NP}$-hard language, then $\mathcal{NP} \subseteq \mathcal{PCP}\left(q, r\right)$.

Remark 7.1. We return to the representation of bits as elements of $\{0,1\}$.

Definition 7.1 (`QuadEq`)**.** `QuadEq` is the language of satisfiable quadratic equation systems over $\{0,1\}$. A quadratic equation system with m equations over n variables is represented as follows:

- *Coefficients:* An $m \times (n \times n)$ block matrix A. For each $k \in [m]$, let A_k denote A's kth block.
- *Free variables:* A length m column vector b.
- *Assignments:* A length n column vector v. An assignment v is satisfies (A, b) when $Avv^\top = b$.

Formally, `QuadEq` $:= \left\{(A, b) \,\middle|\, \exists y \quad Ayy^\top = b\right\}$.

Exercise 7.2. Prove that `QuadEq` is $\mathcal{NP}$-complete.

8 Constructing the PCP: from ideal to real

The PCP itself is not too complicated, and could be prescribed without much motivation and have its correctness analyzed fairly easily (given our work so far). However, we build it iteratively, initially making several assumptions on the given proof, and gradually adding tests to remove each of the assumptions. This section contains mostly motivating/intuitive descriptions of the construction, with only two "formal" technical claims which are used in the actual analysis (Section 9).

8.1 Round one: given a Hadamard encoding of an outer square

As much alluded previously, the PCP will include the Hadamard encoding to a satisfying assignment (or rather, its outer square[7]). To see why the Hadamard encoding of (the square of) a satisfying assignment is useful for our local-to-global task, recall the following:

Fact 8.1 (Distance of the Hadamard code)**.** *For all $u \neq v \in \{0,1\}^k$,*

$$\underset{x \leftarrow \{0,1\}^k}{\mathbb{P}}\left[\langle x, u\rangle = \langle x, v\rangle\right] = 1/2$$

Exercise 8.1. For any $c \times d$ matrix A and column vectors v and a of lengths c and c respectively,

$$\langle v, Aa\rangle = \left\langle A^\top v, a\right\rangle$$

[7]The *outer product* of length n vectors u, v is the $n \times n$ matrix $uv^\top$, whose (i,j)th entry is $u_i v_j$. The *outer square* of a vector is its outer product with itself.

An assignment a is satisfying iff $Aaa^\top = b$ iff $\left\langle \cdot, Aaa^\top \right\rangle = \langle \cdot, b\rangle$ $\left\langle A^\top \cdot, aa^\top \right\rangle = \langle \cdot, b\rangle$. So assuming the verifier is given access to the proof $\pi_1 = \left\langle \cdot, aa^\top \right\rangle$, it would test that $aa^\top = b$ by sampling a uniform $v \leftarrow \{0,1\}^m$ and checking that $\pi_1(Av) = \langle v, b\rangle$ (recall that the verifier has unlimited access to A and b). We formalize this below:

Claim 8.1 (Satisfaction). *For all $v \in \{0,1\}^n$*

$$\mathop{\mathbb{P}}_{x\leftarrow\{0,1\}^m}\left[\left\langle Ax, vv^\top\right\rangle = \langle x, b\rangle\right] = \begin{cases} 1 & v \textit{ satisfies } (A,b) \\ 1/2 & \textit{else} \end{cases}$$

Proof. If v satisfies (A,b) iff $Avv^\top = b$. Using Exercise 8.1: the case that v is satisfying is immediate, and if $Avv^\top \neq b$ then using the distance of the Hadamard code

$$\mathop{\mathbb{P}}_{x\leftarrow\{0,1\}^m}\left[\left\langle A^\top x, vv^\top\right\rangle = \langle x, b\rangle\right] = \mathop{\mathbb{P}}_{x\leftarrow\{0,1\}^m}\left[\left\langle x, Avv^\top\right\rangle = \langle x, b\rangle\right] = 1/2$$

□

However, we cannot assume that the given proof is an encoding of an outer square (or a even a codeword at all). That is, the verifier should reject `NO`cases even when the given proof is not an encoding of an outer square.

8.2 Round two: given a Hadamard encoding of some vector and its outer square

So far, the verifier expects the proof to be a Hadamard encoding of a length n^2 message that is an outer square of a length n message (an assignment). Of all Hadamard codewords (for length n^2 messages), only a negligible fraction ($2^{-n(n-1)}$) encode messages that are outer squares of some assignment. To remove this assumption, the verifier now expects not only for the Hadamard encoding for the assignment's outer square, but also for the Hadamard encoding of the assignment itself. So now the proof has two parts: $\pi = (\pi_1, \pi_2)$ where π_1 should be the Hadamard encoding of the assignment, and π_2 is the Hadamard encoding of the assignment's outer square. If π_1 and π_2 are as expected (one encodes the outer square of the other's message), we say they are *consisted.*

Testing the consistency of π_1 and π_2 relies on the following fact:

Fact 8.2. *For any $x, y, v \in \{0,1\}^n$,*

$$\langle x, v\rangle \langle y, v\rangle = \left\langle xy^\top, vv^\top\right\rangle$$

Similarly to the previous test, the verifier simply checks that this equation on random x, y.

Claim 8.2 (Consistency). *For all $v \in \{0,1\}^n$ and $\nu \in \{0,1\}^{n^2}$*

$$\mathop{\mathbb{P}}_{x,y\leftarrow\{0,1\}^n}\left[\langle x, v\rangle \langle y, v\rangle = \left\langle xy^\top, \nu\right\rangle\right] = \begin{cases} 1 & \nu = vv^\top \\ 1/2 & \nu \neq vv^\top \end{cases}$$

Proof. If $\nu = vv^\top$ we use Fact 8.2. Otherwise, using Fact 8.2 and the fact that if $x, y \leftarrow \{0,1\}^n$ are uniformly sampled then $xy^\top$ is uniformly distributed in $\{0,1\}^{n^2}$, we have

$$\mathop{\mathbb{P}}_{x,y\leftarrow\{0,1\}^n}\left[\langle x, v\rangle \langle y, v\rangle = \left\langle xy^\top, \nu\right\rangle\right] = \mathop{\mathbb{P}}_{x,y\leftarrow\{0,1\}^n}\left[\left\langle xy^\top, vv^\top\right\rangle = \left\langle xy^\top, \nu\right\rangle\right] = 1/2$$

Where the last equality uses the Hadamard code's distance. □

8.3 Final round: given any proof

The given proof π should be the Hadamard encoding of a satisfying assignment π_1 followed by the Hadamard encoding of its outer square π_2. As long as both are Hadamard codewords of the required lengths, the verifier is complete and sound: it always accepts satisfiable equation systems with their correct proofs (i.e. when the given proof is as in the previous sentence), and rejects unsatisfiable equation systems with high probability. The required modification to make the verifier work in the general setting is, unsurprisingly, using the "test and decode" method (which effectively means adding linearity tests on π_1 and π_2), "reducing" the case of the general proof to the case when the proof consists of two codewords.

9 $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{poly}, O(1))$

Algorithm 9.1 (The Hadamard-based PCP verifier)**.** *The verifier gets equation system (A, b) as explicit input and oracle access to proof $\pi = (\pi_1, \pi_2)$. Letting $\widetilde{\pi_1}$ and $\widetilde{\pi_2}$ be the linear functions closest to π_1 and π_2 respectively, the verifier runs the consistency and satisfactions tests on π_1 and π_2 using the "test and decode" method. Explicitly, the verifier first performs (several trials of) linearity tests on π_1 and π_2. It then performs the consistency and satisfaction tests on $\widetilde{\pi_1}$ and $\widetilde{\pi_2}$ using the local decoding algorithm. That is, when the verifier needs $\widetilde{\pi_1}$ at location $x \in \{0,1\}^n$, $r \leftarrow \{0,1\}^n$ is uniformly sampled and the value $\pi_1(x + r) - \pi_1(r)$ is used (and similarly for $\widetilde{\pi_2}$).*

1. Linearity tests: *Repeat the following $6 > 1/\log(8/(7))$ times: Uniformly sample $x, y \leftarrow \{0,1\}^n$ and $u, v \leftarrow \{0,1\}^{n^2}$ and check that*

$$\pi_1(x) + \pi_1(y) = (\pi_1(x + y)), \quad \pi_2(u) + \pi_2(v) = \pi_2(u + v)$$

2. Consistency test: *Uniformly sample $x, y \leftarrow \{0,1\}^n$ and check that*

$$\widetilde{\pi_1}(x)\,\widetilde{\pi_1}(y) = \widetilde{\pi_2}\left(xy^\top\right)$$

3. Satisfaction test: *Uniformly sample $v \leftarrow \{0,1\}^{n^2}$ and check that*

$$\widetilde{\pi_2}\left(A^\top v\right) = \langle v, b\rangle$$

Theorem 9.1. *The Hadamard-based PCP system is complete and sound. Its verifier makes a constant number of queries to the proof oracle, uses a polynomial number of coins, and runs in polynomial time.*

Proof. Leaving completeness and efficiency to the reader, we prove soundness. Let (A, b) be an unsatisfiable equation system. Recall (from the "test and decode" exercise) that if the linearity tests do not reject (which would cause the verifier to reject), then the two queries made to $\widetilde{\pi_1}$ and $\widetilde{\pi_2}$ (each) using the local decoding algorithm are successful with probability at least $1/2$. That is, with probability at least $1/2$, the verifier perfectly emulates the consistency and satisfaction tests on $\widetilde{\pi_1}$ and $\widetilde{\pi_2}$.

In that case (that all local decoding were successful), let $a \in \{0,1\}^n$ and $\alpha \in \{0,1\}^{n^2}$ such that $\widetilde{\pi_1} = \langle \cdot, a\rangle$ and $\widetilde{\pi_2} = \langle \cdot, \alpha\rangle$. If $\alpha \neq aa^\top$ then the consistency test rejects with probability at least $1/2$. Otherwise, $\alpha = aa^\top$, but a is not a satisfying assignment (since (A, b) are unsatisfiable) and then the satisfaction test rejects with probability at least $1/2$.

All in all, with probability at least $(1/2)^2 = 1/4$, either a linear test rejects, or all local decodings succeeded and then the consistency or satisfaction tests reject. □

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, Cambridge, 2009.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, Cambridge; New York, 2008. OCLC: ocn192050142.

[Gol18] Oded Goldreich. *Introduction to Property Testing.* Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA, 2018.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions.* Cambridge University Press, New York, NY, 2014.