IAS/Park City Mathematics Series
Volume 00, 0000

Probabilistically Checkable Proofs

Madhu Sudan

Scribe: Venkatesan Guruswami

Acknowledgments

Mostly, I would like to thank Venkatesan Guruswami for scribing these notes so diligently (despite the fact that tax laws prevented him from getting the honorarium that he fully deserved!).

Thanks also to Johan Håstad and Jakob Nordström for pointing out many errors and typoes.

– Madhu Sudan.

LECTURE 1
Introduction to PCPs

1. Overview

Research in the 1990's has led to the following striking theorem: There is a format of writing proofs and a probabilistic method of verifying their validity, such that the verifier needs to reads only 3 bits of the proof (irrespective of the length of the proof) to obtain probabilistic confidence in the correctness of the proof. Specifically, the verifier accepts correct proofs with probability 1 (Completeness) and given any purported "proof" of an incorrect assertions it accepts with probability at most 3/4 (Soundness). In fact, this probability can be made arbitrarily close to 1/2. Furthermore, the proof in the new format is only polynomially longer than the original "classical" proof.[1]

In addition to being a surprising result bridging probability and logic, the above result also turns out to have applications to proving intractability results for finding near-optimal solutions to many NP-hard optimization problems. Our goal in these lectures will be to provide insight into the construction of these proof systems and the associated probabilistic verifiers. We will not pursue the applications to hardness of approximations (i.e., solving optimization problems near-optimally). The interested reader is referred to the survey article of Arora and Lund [**1**] for more information on such consequences. Our specific target will be to describe the main steps that lead to a weaker result (which we call the PCP Theorem) that the complexity class NP has Probabilistically Checkable Proofs in which the verifier uses logarithmic randomness, queries the proof in only $O(1)$ locations, accepts correct proofs with probability 1, and accepts false proofs with probability bounded away from 1 (say $(1-\epsilon)$ for some constant $\epsilon > 0$).[2] We will also outline some of the ingredients that lead to the sharper result described in the opening sentence.

In the first lecture, we will formally define a Probabilistically Checkable Proof (henceforth PCP). We will briefly discuss the history of its definition and the main initial results in this area. We also define the notion of "gap problems" – the NP-hardness of certain gap problems turns out to be equivalent to the existence of PCPs of the type we seek. Our goal thus leads us to the task of establishing NP-hardness of some convenient (and yet interesting) gap problem. To this end we will define a constraint satisfaction problem based on polynomials that we call PCS (for Polynomial Constraint Satisfaction). We will then state an NP-hardness result of a gap version of PCS and two algorithmic results about polynomials. We will then show that putting these ingredients together, we will see how we can build a non-trivial (but not our final) PCP.

---

[1]The result alluded to here is that of Håstad [**20**]. The picky reader may note some minor discrepancies between result as claimed above and the main result of [**20**]. Such a reader is directed to the work of Guruswami et al. [**19**] (a derivative of [**20**]), which certainly achieves all the claimed properties.

[2]This result was proven by [**3, 2**]. Our presentation of even this result will not be complete — the reader is referred to the original articles for full details. However, we do hope to give a fairly detailed overview of the steps involved. It may be pointed out that the presentation here is somewhat different than in the original works.

Looking ahead to future lectures, in the second lecture we will show how to establish the hardness of PCS with a gap; as well as some overview of the algorithmic results for polynomials. This will conclude the first phase of our task — that of establishing a non-trivial PCP construction. In the third lecture, we will launch into a second phase of PCP constructions. We will see how to construct a variety of PCPs with very different parameters using algebraic methods. None of these PCPs will come close to our specific target PCP. However, they give an idea of the nature of the tools that are available and useful to build PCPs. In the fourth and final lecture, we will introduce a non-algebraic tool in the construction of PCPs, specifically a composition theorem for PCPs. We will show how the composition theorem allows us to use the PCPs constructed in the third lecture (or to close variants of the same) and compose them with each other to get a new PCP that has all the desired properties (for our specific target).

2. Definitions and Formal Statement of Results

The central ingredient of a PCP system is the verifier: a probabilistic polynomial time machine with oracle access to a proof π. The primary resources used by the verifier that are of interest to PCP are the amount of randomness used, and the number of bits of π that are queried by the verifier (once the random coins tossed by the verifier are fixed). This leads to the notion of an (r,q)-restricted verifier: For integer valued functions $r(\cdot)$ and $q(\cdot)$, a verifier is said to be (r,q)-restricted if on every input of length n, it tosses at most $r(n)$ coins and queries the proof for at most $q(n)$ bits.

Definition 1. *For integer valued functions $r(\cdot), q(\cdot)$ defined on integers, and functions $c(\cdot), s(\cdot)$, the class $\mathrm{PCP}_{c,s}[r,q]$ consists of all languages L for which there exists a (r,q)-restricted verifier V with the following properties:*

- [COMPLETENESS]: *$x \in L \Rightarrow \exists\ \pi$ s.t $V^{\pi}(x)$ accepts with probability at least c (over the coin tosses of V).*
- [SOUNDNESS]: *$x \notin L \Rightarrow \forall\ \pi\ V^{\pi}(x)$ accepts with probability $< s$ (over the coin tosses of V).*

In this notation the PCP Theorem states that there exists a constant q such that

$$\mathrm{NP} = \mathrm{PCP}_{1,\frac{1}{2}}[O(\log n), q] \ .$$

At this point some explanation of the role and interrelationships of the parameters may be in order. Note that the definition has *four* parameters: c, s, r and q. Of these four, the randomness $(r)$and query (q) parameters are the ones of primary interest. Usually, the other two parameters will be of subordinate interest. In particular, most PCP constructions today set $c = 1$. Such PCPs are said to have *perfect completeness*, so that "correct" proofs are accepted with probability 1. It is sometimes useful to have the extra flexibility of having $c < 1$ as offered by the definition. However, we won't construct any such PCPs in these lectures. so that is one less parameter to worry about. The soundness of a PCP, in turn, is related to the query complexity and the two can be traded of against each other. Standard techniques used for amplication of error in probabilistic algorithms show how soundness may be reduced by increasing the number of queries. On the other hand, the classical reduction from SAT to 3SAT can be employed to reduce the queries to

3, from any constant, while increasing the soundness but preserving boundedness away from one. Thus to simplify our study we may fix the soundness to some fixed value and then try to minimize the randomness and query complexity. Our choice for this value will be $s = \frac{1}{2}$. When we omit subscripts in the notation PCP$[r, q]$, it is implied that $c = 1$ and $s = \frac{1}{2}$. Finally, we remark on a parameter that we seem to have omitted in the definition, namely the size of the proof. While some papers in the literature study this parameter explicitly, we don't do so here. Instead we let this parameter be captured implicitly by the other parameters. Note that a (r, q)-restricted verifier can make at most 2^{r+q} distinct queries to the proof, and thus the proof size need not be larger than 2^{r+q}. Thus the randomness complexity and query complexity implicitly capture the size of the proof required by a PCP verifier, and we will be satisfied with studying this rough upper bound.

2.1. Some History of Definitions

The definition of PCP actually evolved over a series of surprising developments in the late 80s and early 90s. The notion of checking proofs in a probabilistic sense (where the verification process is allowed to err with small probability) dates back to the seminal work of Goldwasser, Micali and Rackoff [**18**] and Babai [**4**] on Interactive Proofs (IP). In the IP proof system, a probabilistic verifier interacts with a prover who wishes to convince the verifier that some assertion is true. The model of the interactive proofs evolved over time, partly motivated by efforts to understand the model better. One such model was that of "multi-prover interactive proof systems" (MIP) introduced by Ben-Or, Goldwasser, Kilian and Wigderson [**12**]. In this model, a single verifier interacts with multiple provers to verify a given assertion. The MIP proof systems influenced the development of PCPs in two significant ways. On the one hand, many technical results about PCPs go through MIP proof systems, in essential ways. More important to our current context, it led to the definition of the notion of the PCP verifier (though it was not so named then), i.e., a probabilistic verifier with access to an oracle. This notion originated in the work of Fortnow, Rompel and Sipser [**16**] as part of an effort to understand the complexity of MIP proof systems.

All the above works did not place any explicit restrictions on the resources used by the verifier, except the minimal one that it run in (probabilistic) polynomial time. Focus on the efficiency of the verification process started with the work of Babai, Fortnow, Levin and Szegedy [**5**]. Their work focussed on the *computation time* of the verifier and the *size* of the proof. They defined the notion of transparent or holographic proofs, which are proofs that can be checked very efficiently (in polylogarithmic time). The resources of focus in Definition 1 were highlighted by the seminal paper of Feige, Goldwasser, Lovász, Safra and Szegedy [**14**]. Feige et al. established an astonishing connection between probabilistic proof systems for NP and the hardness of approximate solutions to the MAX CLIQUE problem. It became evident from their work that the randomness and query complexity of proof systems were parameters of central interest to inapproximability. However, their work did not abstract a definition of the complexity class PCP. Such a definition was finally abstracted in the work of Arora and Safra [**3**]. Their work explicitly defines the two resources: randomness and query complexity; and maintains them as parameters (rather than placing absolute bounds on them), reflecting the importance of the two resources and the very distinguishable impact that they tend to have on the verification capabilities of the PCP.

2.2. History of Results

The sequence of results culminating in the PCP Theorem is a long one. We will attempt to give a bird's eye view of this history, presenting some of the landmark results. We break this history into four phases.

Phase 0. Some properties of PCPs follow immediately from their definition. These properties, typically attributed to folklore, include results such as NP $=$ PCP$\left[0, \text{poly}(n)\right]$. This is the case because, for any language $L \in$ NP, the verifier can deterministically read the entire polynomial size witness of the membership of $x \in L$ and then choose to accept or reject. It is also not hard to see that NP $=$ PCP$\left[\log n, \text{poly}(n)\right]$. In particular, the the containment PCP$\left[\log n, \text{poly}(n)\right] \subseteq$ NP is obtained as follows:

> Given a PCP verifier V, tossing $r = O(\log n)$ coins and querying (possibly non-adaptively) poly(n) queries, a non-deterministic Turing machine can determine if it accepts with probability 1, by guessing, for each random string, the sequence of queries made and the answers receieved, and then by verifying in polynomial time the following two conditions (1) On each random string verify that the sequence of guessed queries is consistent with V's actions and the responses received lead to acceptance by V. (2) For every pair of random strings identical queries lead to same response.

Thus a little bit of randomness does not increase the power of the PCP verifiers in terms of the languages for which they can verify membership. However it does allow them to be significantly more efficient. (A collection of these and other such folklore results about PCPs may be found in [**9**].)

Phase 1. The first non-trivial result on PCPs did not talk about the class NP but rather about the class NEXP. This result, due to Babai, Fortnow, and Lund [**6**], showed that NEXP $=$ PCP[poly(n), poly(n)]. Note that the traditional verifier of NEXP languages looks at a proof in exponentially many places, while the PCP verifier is only allowed to look at it in polynomially many places. Thus this landmark result reduced the number of queries by a poly-logarithmic amount by using the power of randomness. Subsequently, scaling this result down to NP, Babai, Fortnow, Levin and Szegedy [**5**], NP $\subseteq$ PCP$\left[\text{poly}\ \log n, \text{poly}\ \log n\right]$. The result of [**5**] actually got extremely small blowups, nearly linear, in proof size too, though the implicit bound promised by examining the randomness and query complexity is not even polynomially bounded. The next improvement in the parameters was brought about by Feige et al. who improved the result to NP $\subseteq$ PCP$\left[\log n \log\log n, \log n \log\log n\right]$.

The good news about results in this phase was that they reduce the number of queries made by the verifier by a poly-logarithmic amount (from poly(n) to poly $\log n$), a result that was completely unexpected at the time. However the bad news, is that the randomness and query complexities were still super-logarithmic and hence the above containment are not equalities and thus these do not give *characterizations* of NP in terms of (non-trivial) PCP classes.[3]

Phase 2. The first exact characterization of NP came in the work of Arora and Safra [**3**] who showed that NP $=$ PCP$\big[O(\log n), o(\log n)\big]$. This work also introduced the powerful idea of *recursive composition of proofs* which played a critical role in their and all subsequent improvements to PCP constructions. The PCP Theorem itself (i.e., NP $=$ PCP$\big[O(\log n), O(1)\big]$) was proved by Arora, Lund, Motwani, Sudan and Szegedy [**2**].[4]

As in the results of Phase 1, the results of Phase 2 were startling surprises. The query complexity is independent of the proof size! And both parameters can be reduced to functions which were within constant factors away from the smallest amount conceivable.[5] However these were not yet the ultimate possible PCP results. Specifically, they were not *tight* in either the randomness complexity (or equivalently the proof size) or the query complexity.

Phase 3. Examination of the (non-asymptotic) tightness of the parameters of the PCP theorem was initiated by Bellare, Goldwasser, Lund and Russell [**10**]. Several intermediate results improved the constants in the parameters [**15, 11, 9**]. Eventually near-tight results which optimize both these parameters (but not simultaneously!) were shown. Specifically:

- Polishchuk and Spielman [**23**] showed that Sat $\in$ PCP$\big[(1+\varepsilon)\log n, O(1)\big]$ for every $\varepsilon > 0$.
- It is a folklore result that the number of queries required in the PCP Theorem is at least 3. Håstad [**20**] proved the tight result that for every $\varepsilon > 0$, NP $=$ PCP$_{1-\varepsilon,\frac{1}{2}}\big[O(\log n), 3\big]$. (Note that this result does not have perfect completeness: a later result in [**19**] shows that NP $=$ PCP$_{1,\frac{1}{2}+\varepsilon}\big[O(\log n), 3\big]$.)

The result of Håstad, once again, was a startling development. A folklore result shows that any PCP for an NP-complete language must use $q \geq 3$ to attain perfect completeness. It was also believed that such a PCP could not have soundness $s \leq \frac{1}{2}$ (though this was not proven till much later). Work prior to Håstad's however were far from show that any $s > 1/2$ could be achieved with $q = 3$. In fact, if anything, the belief in days just prior to Håstad's works tended to the conjecture that PCP$_{1,s}[O(\log n), 3]$ may be contained in P for some $s > 1/2$. These beliefs were bolstered by the strong algorithmic techniques, based on "semidefinite programming", introduced in the work of Goemans and Williamson [**17**]. Håstad's results thus brought about (yet another) unexpected settlement of these conjectures. Subsequently, Karloff and Zwick [**22**] used semidefinite programming methods to show the optimality of Håstad's results by showing that PCP$_{1,1/2}[O(\log n), 3] =$ P. Our lectures will unfortunately not be able to go into this phase of developments in the constructions of PCPs; however, we will attempt to provide pointers to this in the concluding lecture.

---

[3]Actually a careful analysis of the protocol in [**5**] shows that the randomness can be made logarithmic; a fact that is related to the fact that the proof size can be made $n^{1+\varepsilon}$ for arbitrarily small $\varepsilon > 0$.

[4]More formally, by a statement like NP $=$ PCP$[O(\log n), O(1)]$, we mean the following: $\exists c_q$ such that $\forall L \in$ NP, $\exists c_r$ such that $L \in$ PCP$[c_r \cdot \log n, c_q]$.

[5]That $\Omega(1)$ queries are required is clear, and a result in [**3**] shows that if NP $\subseteq$ PCP$[o(\log n), o(\log n)]$ then NP $=$ P.

3. Broad Skeleton of the proof

We now move towards the proof of the PCP theorem. The proof that we present will roughly follow the historical path to the proof. We will start by proving a statement similar in spirit to the principal results of Phase 1. Namely, we will first prove (modulo some technical theorems that we will only state) NP $=$ PCP$\big[O(\log n), \text{poly}\log n\big]$. Two fundamental techniques that will be used in its proof are Arithmetization and Low-degree testing. This will occupy the first two lectures. We will then take some digressions. The first one will take us into MIPs and show how the poly $\log n$ queries in the above PCP can be "aggregated" so that the verifier needs to read only $O(1)$ locations of the proof (and receive poly $\log n$ size answers from each location). This will be very useful for us in the final step(s) when we will apply proof composition to reduce the number of queries down to $O(1)$. As a second digression we will show a new PCP verifier for NP that makes only $O(1)$ queries (thus is very good from this perspective) but uses poly(n) randomness (and hence results in exponential sized proofs). Finally, in the final lecture, we will sketch how to prove the PCP Theorem itself by applying the idea of proof composition to the MIP system and this verifier, and show NP $=$ PCP$[O(\log n), O(1)]$.

4. Gap Problems and Polynomial Constraint Satisfaction

4.1. Constraint Satisfaction Problems

Constraint satisfaction problems are a special category of optimization problems that arise naturally in the study of PCP. An instance of the problem consists of a collection of constraints on some variables that take values from some set $[B] = \{1, \ldots, B\}$. The goal is to find an assignment to the variables that maximizes the number of satisfied constraints. More formally, an instance of Max w-CSP(B) consists of n B-ary variables $V = \{x_1, \ldots, x_n\}$ and t w-ary constraints $C_1, \ldots, C_t$ defined on subsets of V of size w. The goal is to find an assignment $a_1, \ldots, a_n \in B$ to the variables V that maximizes the number of satisfied constraints. A well-known example of a constraint satisfaction problem is Max 3-SAT where $w = 3$, $B = 2$ and the constraints are of the form $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ where each ℓ_{i_j} is either x_{i_j} or $\bar{x}_{i_j}$.)

As mentioned earlier the Constraints Satisfaction Problems (henceforth, CSPs) arise naturally in the study of PCP. Informally, PCP$[r, q]$ "corresponds" to Max w-CSP(2) with appropriate relation between the parameters. Roughly, the bits of the proof correspond to the variables (which is why $B = 2$). Each condition checked by the verifier corresponds to a constraint (thus the number of constraints is $t = 2^r$, assuming the verifier is non-adaptive in its queries). The number of queries q equals the "width" w of the CSP. Finally, the acceptance probability of the verifier on a proof equals the fraction of satisfied constraints in the associated assignment to the variables. Thus computing (or even approximating) the maximum number of satisfiable constraints amounts to answering the question: Is the verifier's acceptance probability greater than the completeness, or not? To formally, study the correspondence one needs to work with the notion of gapped problems.

4.2. Gap problems

When dealing with hardness of approximations, it is useful to formulate optimization problems as decision problems with "gaps" associated with them. Gap problems fall into the more general class of "promise" problems whose instances are partitioned into disjoint YES, NO and Don't Care sets. The computational question associated with such a problem is that of deciding whether a given instance is a YES or a NO instance under the promise that the given instance is either a YES instance or a NO instance. (In particular, any answer on an instance from the Don't care set is acceptable.) For CSPs, the associated gap problem, called Gap w-$\mathrm{CSP}_{c,s}(B)$ where $s \leq c$, is the following:

YES instances: $\exists$ assignment that satisfies at least c fraction of the constraints.

NO instance: No assignment satisfies s fraction of the constraints.

The correspondence between PCP and CSP sketched above implies the following which we leave as an (instructive) exercise:

Lemma 1 (Exercise)**.** $\mathrm{NP} = \mathrm{PCP}_{1,1-\varepsilon}\big[O(\log n), 3\big]$ *if and only if Gap* 3-$CSP_{1,1-\varepsilon}(2)$ *is NP-hard.*

(In proving the above, assume that NP-hardness is shown via a many-one reduction from a standard NP-complete problem such as SAT.)

4.3. Polynomial Constraint Satisfaction

From the previous section, to construct PCPs we need to prove NP-hardness of certain gap problems. But then this is only a restatement of the question, and to prove NP-hardness of a gap problem, we need a CSP whose constraints are "robust" in the sense that either all of them can be satisfied or at most a small fraction of them can be satisfied. Low-degree polynomials (over fields) have such a robustness property: if they are zero at "many" places, then are in fact zero everywhere. We now define a CSP called Polynomial Constraint Satisfaction (henceforth referred to as PCS).

Consider a Max w-$\mathrm{CSP}(B)$ problem where $B = \mathbb{F}$ is a finite field and the number of variables $n = |\mathbb{F}|^m$ for some integer m. Thus assignments to the variables can be viewed as functions $f : \mathbb{F}^m \to \mathbb{F}$. The PCS problem is obtained by restricting the assignments f to be some polynomial of (total) degree at most d over $\mathbb{F}$. The formal definition, formulated as a gap problem follows:

Polynomial Constraint Satisfiability Gap $\mathrm{PCS}_{1,\varepsilon}(t, m, w, s, d, q)$:

<u>Instance</u>: Integer valued functions m, w, s, d, q; Finite field $\mathbb{F}$ with $|\mathbb{F}| = q(t)$; Constraints $\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_t$ with each $\tilde{C}_j = \big(C_j;\ \langle x_1^{(j)}, \ldots, x_{w(t)}^{(j)}\rangle \in \mathbb{F}^{m(t)}\big)$ where each $C_j : \mathbb{F}^{w(t)} \to \{0,1\}$ is a $w(t)$-ary constraint over $\mathbb{F}$ that can be computed by a size $s(t)$ algebraic circuit).

<u>YES instances</u>: $\exists$ a degree $d(t)$ polynomial $p : \mathbb{F}^{m(t)} \to \mathbb{F}$ such that for all $j \in \{1, 2, \ldots, t\}$, $C_j\big(p(x_1^{(j)}), \ldots, p(x_{w(t)}^{(j)})\big) = 0$.

<u>NO instances</u>: $\forall$ degree $d(t)$ polynomials $p : \mathbb{F}^{m(t)} \to \mathbb{F}$, the number of $j \in \{1, 2, \ldots, t\}$ such that $C_j\big(p(x_1^{(j)}), \ldots, p(x_{w(t)}^{(j)})\big) = 0$ is less than εt.

For notational convenience we will often omit the parameter t and refer to $m(t), w(t), s(t), d(t), q(t)$ as simply m, w, d, q.

4.4. Hardness of Gap-PCS

The following Lemma (which will be proved in the next Lecture) shows that a Gap version of the PCS problem is NP-hard and thus forms the stepping stone for our PCP constructions.

Lemma 2. *For all constants $\varepsilon > 0$, Gap-PCS$_{1,\varepsilon}(m, w, s, d, q)$ is NP-hard, for $w, s, d, q = \text{poly}\log t$ and $m = O\left(\frac{\log t}{\log\log t}\right)$.*

First note all the good things which the above Lemma gives us. To begin with, we have a gap! Also by the choice of parameters in the NP-hardness, we have $|\mathbb{F}|^m = \text{poly}(t)$ and thus the table of values of f is a reasonable proof to ask the prover to provide. Also the verifier can just pick a random one of the t constraints (which takes only $\log t$ randomness), reading the corresponding $w = \text{poly}\log t$ locations from the table for f and verify that the constraint is satisfied in time $\text{poly}(s(t)) = \text{poly}\log t$. Thus by Lemma 1 we seem to have our first non-trivial PCP characterization (namely NP $\subseteq$ PCP$[O(\log), \text{poly}\log]$). There is a caveat, however; namely the gap (and hence the soundness of the PCP) is guaranteed only when f is restricted to a degree d polynomial, and there is no guarantee that the prover will oblige by conforming to this restriction. Thus we need an efficient way to enforce this low-degree restriction on f which is given by low-degree tests.

5. Low-degree Testing

Ideally, we would like a low-degree test to have the following specification:

Given: $d \in \mathbb{Z}^+$; and oracle $f : \mathbb{F}^m \to \mathbb{F}$

Task: Verify that f is a degree $\leq d$ polynomial in time $\text{poly}(m, d)$; i.e.,

Completeness: If $\deg(f) \leq d$ then accept with probability 1.

Soundness: If $\deg(f) > d$ then reject with high probability.

The above, however, is not possible, since, for every $a \in \mathbb{F}^m$, one can have an f which disagrees with a degree d polynomial at $a \in \mathbb{F}^m$ and agrees with p everywhere else, and thus will pass any test that only queries f at $\text{poly}(m, d)$ places with high probability. We thus need to relax the soundness condition.

Definition 2. *Functions $f, g : \mathbb{F}^m \to \mathbb{F}$ are said to be δ-*close *if $\mathbf{Pr}_x\left[f(x) \neq g(x)\right] \leq \delta$ when x is drawn uniformly at random from $\mathbb{F}^m$.*

Low-degree Test (revised definition):

Given: $\delta > 0$, $d \in \mathbb{Z}^+$; and oracle $f : \mathbb{F}^m \to \mathbb{F}$

Task: Verify that f is close to a degree $\leq d$ polynomial; i.e.,

Completeness: If $\deg(f) \leq d$ then accept with probability 1.

Soundness: Reject with high probability if f is **not δ-close** to any degree $\leq d$ polynomial.

The following result from [**2**] building upon the previous analyses in Rubinfeld and Sudan [**25**] and Arora and Safra [**3**], shows that very efficient low-degree testers do indeed exist. The proof of this result is complicated and we will not delve into it here. We will describe the testing algorithm fully in the second lecture. The interested reader can find all details of the proof in [**2**] and the references cited therein.

Lemma 3 ([**2**]). *There exists a* $\delta_0 > 0$ *such that for every* $\delta < \delta_0$ *there exists a probabilistic solution to the low-degree test that has running time* $\mathrm{poly}(m, d, \frac{1}{\delta})$ *and that tosses* $O(m \log |\mathbb{F}|)$ *random coins.*

6. Self-correction of polynomials

For the choice of parameters in the hardness result of Lemma 2, it follows that the low-degree test of Lemma 3 uses $O(\log t)$ randomness and makes $\mathrm{poly} \log t$ queries to the oracle f. However the gap between the completeness and the soundness of the low-degree test still leaves us with a problematic situation: What to do if the prover provides as proof, a function that is δ-close to a degree d polynomial, which satisfies most constraints? In this case, we get around the problem by testing if the degree d polynomial g that is δ-close to the oracle f satisfies most constraints. But how can we get our hands an oracle for g? It turns out we can implement such an oracle, probabilistically, using the oracle for f. The self-correction problem formalizes the task at hand; and the subsequent lemma shows how efficiently this problem can be solved.

Self-correction of Multivariate polynomials:

Given: $\delta > 0$; $d \in \mathbb{Z}^+$; $x \in \mathbb{F}^m$; oracle $f : \mathbb{F}^m \to \mathbb{F}$ such that f is δ-close to some degree d polynomial p. (We assume $\delta < \frac{d}{2|\mathbb{F}|}$ so that a polynomial p that is δ-close to f, if one exists, is *unique*.)

Task: Compute $p(x)$.

The following result from [**7**] shows the existence of randomized self-correctors for multivariate polynomials.

Lemma 4. *There exists a randomized algorithm that solves the self-correction problem that runs in time* $\mathrm{poly}(m, d, \frac{1}{\delta})$ *and tosses* $O(m \log |\mathbb{F}|)$ *random coins, and outputs the right answer (for every* x*) with probability at least* $(1 - \varepsilon)$ *provided* $\delta < \min\{\frac{d}{2|\mathbb{F}|}, \frac{\varepsilon}{d+1}\}$.

The proof of the above lemma is not difficult and will be presented in the next lecture. For now we just assume this lemma for a fact and move towards the PCP that gives us the result of Phase 1.

7. Obtaining a non-trivial PCP

Armed with Lemmas 2, 3 and 4 we can now give our first PCP verifier that works as follows. Let $L \in \mathrm{NP}$. Given x purportedly in L, the verifier computes (in polynomial time) an instance ϕ of Gap-PCS as guaranteed in the NP-hardness result of Lemma 2. The prover supplies an oracle for an assignment $f : \mathbb{F}^m \to \mathbb{F}$ (plus other auxiliary information which may be used by the low-degree test). The verification process proceeds as follows:

1. Run the Low-degree test from Lemma 3 on f. Reject if the test rejects.
2. Pick a random constraint C of ϕ and verify that Self-correct(f) satisfies C (where the algorithm Self-correct is obtained from Lemma 4). Reject if not.
3. Accept otherwise.

From the statements of Lemmas 2, 3 and 4, it follows that the above verifier queries $\mathrm{poly} \log |x|$ bits in the proof, tosses $O(\log |x|)$ random coins, has perfect completeness $c = 1$ and soundness $s \ll \frac{1}{2}$. We thus have our first step:

Theorem 1. $\mathrm{NP} = \mathrm{PCP}\big[O(\log n), \mathrm{poly}\log n\big]$.

The agenda for the next lecture is to give further details on the proofs of Lemmas 2-4 on the NP-hardness of Gap-PCS.

LECTURE 2
NP-Hardness of PCS

In this lecture we will set out and prove the NP-hardness of Gap-PCS (Lemma 2 from previous lecture) and present a self-corrector for multivariate polynomials (Lemma 4 from previous lecture) and there by complete Phase I of the proof; i.e., establish NP $=$ PCP$[O(\log n), \operatorname{poly}\log n]$. (For the other result, Lemma 3, on low-degree tests, we will only present a test and and take its analysis on faith.)

1. Multivariate polynomials

All of our lemmas seem to involve polynomials, while our original goal of constructing PCPs (seemingly) had nothing to do with polynomials. Before, plunging into the proofs of the lemmas, it may be worth our while to see why polynomials arise naturally in this context.

We first note a robustness property that proofs in the PCP format seem to have. Specifically, if we take a valid proof (accepted with probability 1) in the 3-query PCP of, say Håstad, the proof has the property that when 1% of the bits are flipped at random then its acceptance probability is still at least 97%. Thus PCP proofs are special in that they retain the power to convince a verifier even when a reasonably large fraction of their bits are flipped, completely at random. A natural question to ask is: How does the proof develop this resilience to error? Turns out that a previous context in which similar resilience to error was explored was in the context of information transmission over noisy channels. This research led to the development of error-correcting codes. Informally, an error-correcting code consists of an encoding function that maps a small string (message) into a large one (codeword) such that flipping a few bits of the codeword, still allows for recovery of the message. Our strategy to endow the PCP proofs with redundancy will exploit the theory directly. We will simply encode traditional proofs using well-known error-correcting encodings and this will bring about the necessary resilience. However an arbitrary error-correcting code will not suffice for our purposes. We will use a special construction of error-correcting codes: those obtained by employing (multivariate) polynomials over finite fields.

Polynomials (over a field) are known to have excellent error-correction properties (in addition to their nice algebraic structure). As an example, consider the following encoding of a string $a_1, \ldots, a_n \in \{0,1\}^n$. Pick a finite field $\mathbb{F}$ of size about n^2

and let f be a polynomial of degree less than n such that $f(1) = a_1, \cdots, f(n) = a_n$[1]. Note that such a polynomial does indeed exist, and can be found be interpolation. Then $\langle f(x)\rangle_{x\in\mathbb{F}}$ is a redundant encoding of $a_1, \ldots, a_n$ in the following sense: Given the value of f at any subset of $\mathbb{F}$ of size n, we can interpolate to find f and thus the coefficients $a_1, \ldots, a_n$. The original string can be reconstructed even if $\frac{|\mathbb{F}-n|}{2}$ of the symbols in its encoding are in error.

Codes based on univariate polynomials gives robustness against a huge fraction of errors and is extremely efficient in this sense. For our purposes the primary disadvantage of these codes is that to encode an n-bit string, it needs degree $\Omega(n)$. In particular, this implies that any version of the Low-degree test would need to query the value of any function f at $\Omega(n)$ places at the very least, before being able to conclude that the given function is not a degree n polynomial.

To get better low-degree tests, one needs to find functions whose algebraic degree is somehow smaller than the number of degrees of freedom that the function exhibits. Bivariate polynomials already exhibit better tradeoffs. For example we may pick a field $\mathbb{F}$ of cardinality $\approx n$ and pick a polynomial f in two variables x and y of degree at most $\sqrt{n}$ in each such that the value of f at the points $\{(i,j) | 0 \le i \le \sqrt{n}\}$ correspond to the values $a_1, \ldots, a_n \in \{0,1\}$. (Again, one needs to verify that such an f exists, and can be found. This task is left to the reader as an exercise.) Now the sequence $\langle f(x,y)\rangle_{x,y\in\mathbb{F}}$ forms another redundant encoding of the string $a_1, \ldots, a_n$.

We can now generalize this idea further to m-variate polynomials over a large enough field $\mathbb{F}$ as follows: Pick a subset $H \subseteq \mathbb{F}$ of size $n^{1/m}$ so that the information $a_1, \ldots, a_n$ can be viewed as a function $a : H^m \to \{0,1\}$. In this case, it can be shown (again left as an exercise to the reader) that there exists an m-variate polynomial f of degree less than $|H|$ in each variable such that $f(x) = a(x)$ for each $x \in H^m$. Now encode a by $\langle f(x)\rangle_{x\in\mathbb{F}^m}$. This construction will be invoked often in the sequel, and it will be useful to give it a name — we call f the *low-degree extension* of a. The redundancy of this encoding follows by the following lemma, referred to in the computer science literature as the Schwartz-Zippel lemma.

Lemma 5. *For every integer m, d, field $\mathbb{F}$ and finite subset $S \subseteq \mathbb{F}$, if $P : \mathbb{F}^m \to \mathbb{F}$ is a non-zero polynomial of total degree at most d, then the probability that $P(x) = 0$, when x is chosen uniformly at random from S^m, is at most $d/|S|$.*

The lemma is easy to prove by induction on the number of variables and we skip the proof.

For our application to PCS, we will pick $m(n) = O(\frac{\log n}{\log\log n})$ and $|H| = \text{poly}\log n$. Thus the degree of the low-degree extension of a is $\text{poly}\log n$ (which is good) and we can work with a field $\mathbb{F}$ of size $\text{poly}\log n$ and still have $|\mathbb{F}|^m = \text{poly}(n)$ so that the size of encoding is polynomial in n.

[1] Note that we are abusing notation by using integers to represent elements of the finite field. We do so only for notational convenience.

2. Hardness of Gap-PCS

2.1. Arithmetizing 3-SAT

We will establish the NP-hardness of Gap-PCS by reducing from 3-SAT. We begin by describing the powerful idea of arithmetizing 3-SAT which is at the heart of the reduction.

An instance ϕ of 3-SAT consists of n variables and t clauses $C_1, \ldots, C_t$ where each clause C_j is of the form $[\ x_{i_1} = b_1$ or $x_{i_2} = b_2$ or $x_{i_3} = b_3\ ]$ where each $b_j \in \{0,1\}$. We find it convenient to view ϕ as an indicator function $\phi : \{1,2,\ldots,n\}^3 \times \{0,1\}^3 \to \{0,1\}$ where $\phi(i_1,i_2,i_3,b_1,b_2,b_3) = 1$ exactly if the clause $[\ x_{i_1} = b_1$ or $x_{i_2} = b_2$ or $x_{i_3} = b_3\ ]$ is present in the instance ϕ.

To arithmetize ϕ, we begin by picking h, m where $h = \operatorname{poly}\log n$ and $m = O(\log n / \log\log n)$ such that $h^m = n$. Now, set $H = \{1,2,\ldots,h\}$ and identify $\{1,\ldots,n\}$ with H^m in some canonical way. Extending $\{0,1\}$ to H, the instance ϕ can be viewed as a function $\phi : H^\ell \to \{0,1\}$ where $\ell = 3m+3$ (we set $\phi(\cdots) = 0$ if the arguments do not make sense).

In this language, 3-SAT can be restated as follows: we want an "assignment" $a : H^m \to \{0,1\}$ such that $\forall i_1, i_2, i_3 \in H^m$ and $\forall b_1, b_2, b_3 \in H$,

$$\phi(i_1,i_2,i_3,b_1,b_2,b_3) = 0 \text{ or } a(i_1) = b_1 \text{ or } a(i_2) = b_2 \text{ or } a(i_3) = b_3 \ .$$

Let $\mathbb{F}$ be a field that contains H and let $\hat{\phi}$ and A be low-degree extensions of ϕ and a respectively. Now the "proof" of satisfiability is an m-variate polynomial (of degree h in each variable) $A : \mathbb{F}^m \to \mathbb{F}$ and the goal of the verifier is to check that for all $z = \langle i_1,i_2,i_3,b_1,b_2,b_3\rangle \in H^\ell$,

$$\hat{\phi}(z) \cdot (A(i_1) - b_1) \cdot (A(i_2) - b_2) \cdot (A(i_3) - b_3) = 0 \ . \tag{1}$$

It is easy to see that such an m-variate polynomial A exists iff ϕ is satisfiable. Thus if we consider the instance of the PCS problem, consisting of $t = |H|^\ell$ constraints of the form (1) for every $z \in H^\ell$, we obtain an instance of the PCS problem for which it is NP-hard to decide if all constraints are satisfiable or not. Thus we have the NP-hardness of a PCS problem. However, there is no gap in the number of constraints (1) that can be satisfied.

2.2. Making Constraints Robust

We now show how to make the constraints above robust, i.e., transform them into a different collection in which either all of them can be satisfied, or few can be satisfied. To this end we define an ℓ-variate polynomial $\tilde{P}_0$ as follows: $\forall z = \langle i_1,i_2,i_3,b_1,b_2,b_3\rangle$,

$$\tilde{P}_0(z) \stackrel{\text{def}}{=} \hat{\phi}(z) \cdot (A(i_1) - b_1) \cdot (A(i_2) - b_2) \cdot (A(i_3) - b_3) \ . \tag{2}$$

Since $\hat{\phi}$ and A have degree at most $|H|$ in each variable, $\tilde{P}_0(z)$ is an ℓ-variate polynomial of degree at most $2|H|$ in each variable and thus has (total) degree at most $2\ell|H|$. Let us assume that the prover gives not only the polynomial A, but also a polynomial P_0 (of degree at most $2\ell|H|$) that is supposedly $\tilde{P}_0$. The goal of the verifier is now to check the constraints

1. $(C0)$: $\forall z \in \mathbb{F}^\ell$ $P_0(z) = \tilde{P}_0(z)$ (note that the verifier can efficiently compute $\hat{\phi}(z)$ and thus also $\tilde{P}_0(z)$ once it is given the assignment polynomial A).
2. $(C0')$ $\forall z \in H^\ell$ $P_0(z) = 0$.

Since both P_0 and $\tilde{P}_0$ are low-degree polynomials (they have degree at most $2\ell|H|$), the constraints $(C0)$ are robust (either all of them are satisfied or a small fraction (at most $\frac{2\ell|H|}{|\mathbb{F}|}$: see lemma below) of them are satisfied.

Lemma 6. *If P_0, $\tilde{P}_0$ are degree d polynomials that violate $(C0)$ for some z, then they violate $(C0)$ for at least $\left(1 - \frac{d}{|\mathbb{F}|}\right)$ fraction of the z's.*

Proof: Follows from the Schwartz-Zippel Lemma applied to $P_0 - \tilde{P}_0$ since a degree d polynomial is zero on at most $\frac{d}{|\mathbb{F}|}$ fraction of the domain. ∎

The constraints $(C0')$ are not robust, since it is possible for a degree $2\ell|H|$ polynomial to be zero on all but one point of H^ℓ. Our idea would be to increase the size of the domain on which we would like the polynomial to be zero. Specifically we will define a sequence of (low-degree) polynomials $P_1, P_2, \ldots, P_\ell$ such that $P_1 = 0$ over $F \times H^{\ell-1}$ iff $P_0 = 0$ over H^ℓ, and similarly for $1 < i \leq \ell$, $P_i = 0$ over $F^i \times H^{\ell-i}$ iff $P_{i-1} = 0$ over $F^{i-1} \times H^{\ell-i+1}$. Hence P_ℓ will be identically zero on F^ℓ iff $P_0(z) = 0$ $\forall z \in H^\ell$. Each of these constraints (and in particular $P_\ell(z) = 0$ $\forall z \in F^\ell$ are all robust constraints and this will give us the desired "gap" in the PCS instance.

As a motivation for defining these polynomials, let us first look at an analogous transformation for univariate polynomials. Let $\{h_1, h_2, \ldots, h_{|H|}\}$ be an enumeration of the elements of H. Given a univariate polynomial $p \in \mathbb{F}[X]$, define a polynomial q by:

$$q(y) = \sum_{j=1}^{|H|} p(h_j) y^j \ .$$

Clearly, if $p(h) = 0$ for all $h \in H$, the $q \equiv 0$. Conversely, if $p_{|H} \not\equiv 0$, then q is some non-zero polynomial of degree at most $|H|$ and so is non-zero on at least $|\mathbb{F} \setminus H|$ points. Thus q is identically zero on $\mathbb{F}$ iff p is identically zero on H.

In the multivariate case, we will apply the above transformation, once in each variable. Starting with a polynomial P_0 in formal variables $(x_1, x_2, \ldots, x_\ell)$, we will obtain a sequence of polynomials

$$\begin{array}{l} P_1(y_1, x_2, \ldots, x_\ell) \\ P_2(y_1, y_2, x_3, \ldots, x_\ell) \\ \quad \vdots \\ P_i(y_1, y_2, \ldots, y_i, x_{i+1}, \ldots, x_\ell) \\ \quad \vdots \\ P_\ell(y_1 . y_2, \ldots, y_\ell) \end{array}$$

where each transition from an x-variable to a y-variable follows the scheme described above for univariate polynomials, namely, for $1 \leq i \leq \ell$, define

$$P_i(y_1, \ldots, y_i, x_{i+1}, \ldots, x_\ell) = \sum_{j=1}^{|H|} P_{i-1}(y_1, \ldots, y_{i-1}, h_j, x_{i+1}, \ldots, x_\ell) y_i^j \ . \tag{3}$$

Note that if P_{i-1} has degree d_{i-1}, then the degree d_i of P_i is at most $d_{i-1} + |H|$. Since P_0 has degree at most $2\ell|H|$, the degree of each P_i for $i \in \{0, 1, \ldots, \ell\}$ is clearly at most $3\ell|H|$. By the same reasoning as in the univariate case, we have

$$P_i\ _{|F^i \times H^{\ell-i}} \equiv 0 \Longleftrightarrow P_{i-1}\ _{|F^{i-1} \times H^{\ell-i+1}} \equiv 0 \ .$$

(By our definitions, we have

$$P_\ell(y_1,\ldots,y_\ell) = \sum_{1\le i_1,i_2,\ldots,i_\ell\le|H|} P_0(h_{i_1},\ldots,h_{i_l})y_1^{i_1}\cdots y_\ell^{i_\ell} .$$

and this is another way of verifying that $P_\ell \equiv 0$ on F^ℓ iff P_0 is identically zero on H^ℓ.)

2.3. The Gap-PCS instance

We are now ready to describe the constraints of our Gap-PCS instance. Given a 3-SAT instance ϕ, consider the following (polynomial) constraint satisfaction problem: The required "solution" consists of polynomials $A, P_0, P_1, \ldots, P_\ell$ where A is an m-variate polynomial of degree at most $m|H|$ and $P_0, \ldots, P_\ell$ are ℓ-variate polynomials of degree at most $3\ell|H|$. The "constraints" placed on the polynomials are the following.

For all $z = (z_1, \ldots, z_\ell) \in F^\ell$:

$(C0)$: $P_0(z) = \tilde{P}_0(z)$ where $\tilde{P}_0(z)$ is defined based on ϕ and $A : \mathbb{F}^m \to \mathbb{F}$ as in Equation (2).

For $i = 1, 2, \ldots, \ell$,

(Ci): $P_i(z_1,\ldots,z_i,z_{i+1},\ldots,z_\ell) = \sum_{j=1}^{|H|} P_{i-1}(z_1,\ldots,z_{i-1},h_j,z_{i+1},\ldots,z_\ell)z_i^j$
(the condition from Equation (3) at the point z).

$(C(\ell+1))$: $P_\ell(z) = 0$.

By the "robustness" of all these constraints (see Lemma 6 above), we have the following:

Lemma 7. *If $P_0, \ldots, P_\ell$ and $\tilde{P}_0$ are polynomials of degree at most d, then for each set of $|\mathbb{F}|^\ell$ constraints (Ci), $0 \le i \le \ell+1$, either all of them are satisfied or at most a fraction $(d+|H|)/|\mathbb{F}|$ of them are satisfied.*

Proof: Follows from Lemma 6 since all polynomials involved in the constraints have degree at most $d + |H|$. ∎

Bundling polynomials into a single polynomial. Note that in a PCS instance the "solution" asked for is a single low-degree polynomial, where as in the above we have several polynomials $(A, P_0, \ldots, P_\ell)$ involved in the constraints. There is a simple trick to handle to this: we just require that all the polynomials be "bundled together" and presented as a single degree $D = (3\ell|H| + \ell + 1)$ polynomial $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$ such that for $0 \le i \le \ell$, $Q(i,\cdots) = P_i(\cdots)$ and $Q(\ell+1, \langle z_1,\ldots,z_\ell\rangle) = A(z_1,\ldots,z_m)$. The existence of such a polynomial is guaranteed by the following Lemma:

Lemma 8. *Given polynomials $q_0, \ldots, q_t : \mathbb{F}^\ell \to \mathbb{F}$ over a finite field $\mathbb{F}$ with $|\mathbb{F}| > t$, each of (total) degree at most α, there exists a degree $\alpha+t$ polynomial $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$ such that for $i = 0, 1, \ldots, t$ and all $z \in \mathbb{F}^\ell$, $Q(i, z) = q_i(z)$.*

Proof: For each $i \in \{0, 1, \ldots, t\}$, there is a unique univariate polynomial δ_i of degree t such that

$$\delta_i(v) = \begin{cases} 1 & \text{if } v = i \\ 0 & \text{if } 0 \le v \le t \text{ but } v \ne i. \end{cases}$$

Now define the polynomial Q as

$$Q(v,z) = \sum_{i=0}^{t} \delta_i(v) q_i(z) \ .$$

Clearly $Q(i,\cdots) \equiv q_i(\cdots)$ for each $i \in \{0,1,\ldots,t\}$. ∎

Suppose such a polynomial Q is given (as a solution to the PCS instance constructed from ϕ). We wish to describe the constraints of the PCS instance. First we make explicit the definition of a polynomial P'_0 from Q that will serve the role of $\tilde{P}_0$ from definition (2). For $z = \langle z_1, \ldots, z_\ell \rangle \in \mathbb{F}^\ell$ where $\ell = 3m+3$, $P'(z)$ is defined as:

$$\begin{aligned} P'_0(z) \stackrel{\text{def}}{=} \ & \hat{\phi}(z) \cdot \big(Q(\ell+1, \langle z_1, \ldots, z_m, 0, \ldots, 0\rangle) - z_{3m+1}\big) \\ & \cdot \big(Q(\ell+1, \langle z_{m+1}, \ldots, z_{2m}, 0, \ldots, 0\rangle) - z_{3m+2}\big) \\ & \cdot \big(Q(\ell+1, \langle z_{2m+1}, \ldots, z_{3m}, 0, \ldots, 0\rangle) - z_{3m+3}\big) \end{aligned} \tag{4}$$

Note that P'_0 has total degree at most $10\ell|H| + 3\ell + 3 < 11\ell|H|$.

Summarizing the reduction from SAT to PCS. We are now ready to summarize the reduction $T_{\text{3SAT}\to\text{PCS}}$ which maps instances of 3SAT to PCS: Given an instance ϕ of 3SAT, the reducing algorithm sets $m = \frac{\log n}{\log logn}$ and sets $h = n^{1/m}$, and $\ell = 3m+3$. It then picks a field $\mathbb{F}$ of sufficiently large size, say $q \geq h^3$. (The choice of 3 in the exponent is somewhat arbitrary. We will need $q \gg mh$ and $q \geq h^3$ suffices for our purpose.) It then computes the function $\hat{\phi} : \mathbb{F}^\ell \to \mathbb{F}$, and using this, it generates $t = |\mathbb{F}|^\ell$ constraints $(C)(z)$, one for every $z \in \mathbb{F}^\ell$. The constraint for z is:

$$(C)(z) = \bigwedge_{i=0}^{\ell+1} (Ci)(z)$$

where (Ci) are the constraints described earlier in this section. The main exception is that these constraints are defined over a single polynomial $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$, and thus every occurence of $P_i(\cdot)$, $0 \leq i \leq \ell+1$ is replaced with $Q(i,\cdot)$. Similarly instead of the polynomial $\tilde{P}_0$ one uses the polynomial P'_0 defined in Equation (4). All polynomials involved in constraints $(C)(z)$ have degree at most $11\ell|H|$, and hence we get by Lemma 6 that, for any degree D polynomial Q, either all the constraints $(C)(z)$ are satisfied or at most a fraction $11\ell|H|/|\mathbb{F}|$ of the constraints are satisfied. By choice of $|\mathbb{F}|$ this fraction is a $o(1)$ function and thus is smaller than ϵ, for any $\epsilon > 0$, for sufficiently large n.

2.4. The hardness result

From the discussion in the preceding paragraph, we can now conclude:

Lemma 9. *For every $\epsilon > 0$, the reduction $T_{\text{3SAT}\to\text{PCS}}$ maps an instance ϕ to an instance of PCS with $m = O(\log n/\log\log n)$ and $w, d, q = \text{poly}\log n$ such that the following conditions are satisfied:*

- **Completeness:** *If ϕ is satisfiable, then there exists a polynomial Q of degree at most d that satisfies all the constraints.*
- **Soundness:** *If there exists a polynomial Q of degree at most D that satisfies more than an ϵ-fraction of the constraints, then ϕ is satisfiable.*

Proof: The completeness is clear since we can just take Q to be the polynomial such that $Q(\ell+1,\cdot) = A$, $Q(0,\cdot) \equiv \tilde{P}_0(\cdot)$ (where $\tilde{P}_0$ is defined in Equation (2)) and $Q(i,\cdot) = P_i(\cdot)$ (where P_i is defined as in Equation (3)) for $1 \leq i \leq \ell$. For the soundness, we know by the discussion at the end of the previous subsection, that if more than an ϵ-fraction of the constraints are satisfied, then in fact all of them are satisfied. This in turn implies that $\tilde{P}_0(\cdot) = Q(0,\cdot)$ is identically zero on H^ℓ, which implies that the assignment $A \stackrel{\text{def}}{=} Q(\ell+1,\cdot)$ satisfies ϕ. ∎

Note that by the choice of the parameters, we have $m = O(\log n/\log\log n)$ and $w, d, q = \text{poly}\log n$ as required. Finally, for each $z \in \mathbb{F}^{l}$, the constraint $(C)(z)$ can be checked in polylogarithmic time. We have thus proved the first of the lemmas from last lecture that we set out to prove:

Lemma 2: *For all constants $\varepsilon > 0$, Gap-PCS$_{1,\varepsilon}(m, w, s, d, q)$ is NP-hard, for $w, s, d, q = \text{poly}\log t$ and $m = O\left(\frac{\log t}{\log\log t}\right)$.*

3. Low-degree Testing

Recall the following Lemma from the previous lecture:

Lemma 3: *There exists a $\delta_0 > 0$ such that for every $\delta < \delta_0$ there exists a probabilistic solution to the low-degree test that has running time* $\text{poly}(m, d, \frac{1}{\delta})$ *and that tosses $O(m \log |\mathbb{F}|)$ random coins.*

We will not be able to prove the above lemma, but we will present the testing algorithm which has the properties claimed in the lemma. The idea behind the test is the following: For $x, y \in \mathbb{F}^m$, define $f_{x,y}(t) = f(x+ty)$ (i.e., $f_{x,y}$ is f restricted to the "line" passing through x with slope y). If f is a degree d polynomial, then for every $x, y \in \mathbb{F}^m$, $f_{x,y}$ is a (univariate) polynomial of degree d, and in fact the converse also holds. This suggests the following test:

Pick random x, y and verify that $f_{x,y}$ is a degree d polynomial.

We in fact consider the following weaker test Low-Deg-Test:

- Pick $x, y \in \mathbb{F}^m$ and $t \in \mathbb{F}$ at random.
- Ask prover for (the at most $(d+1)$) coefficients of the "polynomial" $f_{x,y}$
- Verify that $f_{x,y}(t) = f(x+ty)$.

The following theorem [**25, 3, 2**] shows that the above test indeed satisfies the conditions of Lemma 3.

Theorem 2. *Consider the test* Low-Deg-Test *specified above.*

1. *Easy part: If f is a degree d polynomial, then there exist responses $f_{x,y}$ such that* Low-Deg-Test *always accepts.*
2. *Hard part: There exists a constant $\delta_0 > 0$ such that for all $m, d, \mathbb{F}$, if f is any function such that there exist responses $f_{x,y}$ that make* Low-Deg-test *reject with probability $\delta \leq \delta_0$, then f is 2δ-close to some degree d polynomial.*

4. Self-correction

We now move to the third and final component we need to complete our first PCP characterization (NP $=$ PCP$[O(\log n), \text{poly}\log n]$), namely self-correction. Recall the problem definition:

Given: $\delta > 0$; $d \in \mathbb{Z}^+$; $x \in \mathbb{F}^m$; oracle $f : \mathbb{F}^m \to \mathbb{F}$ such that f is δ-close to some degree d polynomial p. (We assume $\delta < \frac{d}{2|\mathbb{F}|}$ so that a polynomial p that is δ-close to f, if one exists, is *unique*.)

Task: Compute $p(x)$.

We will prove:

Lemma 4: *There exists a randomized algorithm that solves the self-correction problem that runs in time* $\mathrm{poly}(m, d, \log \frac{1}{\epsilon})$ *and tosses* $O(m \log |\mathbb{F}|)$ *random coins, and outputs the right answer (for every x) with probability at least* $(1 - \varepsilon)$ *provided* $\delta < \frac{d}{2|\mathbb{F}|}$.

Proof: Consider the following self-correction procedure. Given $x \in \mathbb{F}^m$ and oracle for f which is δ-close to a polynomial p, compute $p(x)$ as follows:

1. Pick $y \in \mathbb{F}^m$ at random.
2. Query $f(x + y)$, $f(x + 2y), \cdots, f(x + (d+1)y)$ and let $b_1, \ldots, b_{d+1}$ be the responses.
3. Find, by interpolation, a degree d (univariate) polynomial h such that $h(i) = b_i$ for $1 \le i \le d + 1$.
4. Output $h(0)$ as the value of $p(x)$.

Note that the algorithm tosses $O(m \log |\mathbb{F}|)$ random coins, probes f in $d + 1$ places and runs in time polynomial in m, d. It remains to prove the correctness of the procedure. If f is a degree d polynomial, then the output is clearly correct. But f is only δ-close to a degree d polynomial p. However, for every i, $1 \le i \le d + 1$, $x + iy$ is a random point in $\mathbb{F}^m$ (we are ignoring the possibility that $y = 0$ here, but this happens with negligible probability). Thus, $\mathbf{Pr}_y[f(x + iy) \neq p(x + iy)] \le \delta$ by the definition of δ-closeness. Hence, by the union bound, $\mathbf{Pr}_y[\exists i, f(x + iy) \neq p(x + iy)] \le (d + 1)\delta$ which is at most ε since $\delta < \varepsilon/(d + 1)$. Thus, with probability at least $(1 - \varepsilon)$, $b_1, \ldots, b_{d+1}$ are the "right" values of $p(x + y), \ldots, p(x + (d + 1)y)$ and thus the interpolation step correctly computes $p(x)$. ∎

This completes the proof of the PCP characterization $\mathrm{NP} = \mathrm{PCP}[O(\log n), \mathrm{poly}\log n]$. (Recall that the easier direction of the containment was already shown in Section 2.2.) This completes Phase 1 of our goals.

LECTURE 3
A couple of digressions

We now move on Phase 2 of the proof of the PCP Theorem. We will approach this phase somewhat tangentially. In this lecture, we will show two results, that will essentially be digressions for now, and then linked to Phase 2 in the final lecture. The first result will be an "MIP" characterization of NP. We will show how the PCP verifier of Phase 1 can be modified into an MIP verifier that "aggregates" the $\operatorname{poly}\log n$ queries of the PCP verifier into a constant number of queries that it will send to multiple (mutually non-interacting) provers that respond with $\operatorname{poly}\log n$ bits each. While the advantage of this modification will be unclear for now, we will exploit this MIP verifier in the final lecture. The second result will give a highly query-efficient PCP verifier for NP: specifically we will prove that $\mathrm{NP} = \mathrm{PCP}[\operatorname{poly}(n), O(1)]$. Note that this verifier just makes a constant number of queries (as is our final goal), however that the randomness used by the verifier is very large.

Part I: Multiprover Interactive Proofs (MIP)

The informal question behind the definition of MIP is the following: What can a probabilistic verifier interacting with p non-communicating provers verify, if allowed to ask one question to each prover? More formally, we have the following definition:

Definition 3. *For an integer p and integer valued functions $r, a : \mathbb{Z}^+ \to \mathbb{Z}^+$, a (p, r, a)-restricted MIP verifier is a probabilistic verifier that tosses $r(n)$ coins, asks one question to each of p provers and receives $a(n)$-bit answers, on inputs of length n.*

We can now define MIP classes similar to PCP classes.

Definition 4. *For an integer p and integer valued functions $r, a : \mathbb{Z}^+ \to \mathbb{Z}^+$, a language L is said to be in $\mathrm{MIP}_{c,s}[p, r, a]$ if there is a (p, r, a)-restricted MIP verifier that checks $x \in L$ with completeness c and soundness s.*

A p-prover MIP is also called a p-prover 1-round protocol, since there is only one round of verifier-prover interaction. A few comments on the MIP model. MIP seems to be a natural model within the context of interactive proofs. It is more restrictive than PCP as $\mathrm{MIP}_{c,s}[p, r, a] \subseteq \mathrm{PCP}_{c,s}[r, pa]$ (since the responses of the

p provers can be written down as one big proof, and the verifier will query $pa(n)$ bits from this proof), and thus good MIP constructions suffice to construct good PCPs. We will now do the opposite and show how to covert a PCP into a MIP (with some loss in parameters), and this will be a central intermediate step in our goal of proving the PCP Theorem.

1. A 3-prover MIP for NP

We will construct a 3-prover MIP from the PCP$[O(\log n), \mathrm{poly} \log n]$ verifier of Phase 1. To do this, let us first recall how the verifier worked (at a high level). The verifier is based on a hard instance of PCS with a "gap". It expects as proof a low-degree polynomial expressed as a table of values $f : \mathbb{F}^m \to \mathbb{F}$, and a "lines oracle" f_{lines} that it uses for performing a low-degree test. Given access to oracles for f and f_{lines}, the verifier worked in two steps:

1. Perform a low-degree test on f.
2. Pick a random constraint of the PCS instance and check it is satisfied by the self-corrected version of the oracle f.

The first step above is already structured as a 2-prover 1-round protocol: The verifier asks one prover for the value of f at a point and a second prover for the coefficients of the polynomial $f_{x,y}$ for a line $\ell_{x,y} = \{x + ty : t \in \mathbb{F}\}$ for some $x, y \in \mathbb{F}^m$. The second step, however, queries the table f in many places, and we somehow need a way to "aggregate" these queries into one "big" query.

1.1. Parallelization: Reconstruction via curves

Suppose we need to query $f : \mathbb{F}^m \to \mathbb{F}$ at w places $x_1, \ldots, x_w$. In this section we will show how to find the value of f at all these points correctly, with high probability, using only a constant number of queries to two provers. This solution will work using the "algebraic" and "randomness" properties of "curves" in m-dimensional space (where all terms in quotes will be explained later). Using such curves, our strategy can be described at a high-level as follows: We will pick a *random curve* C through $x_1, \ldots, x_w$ and ask a third prover for a description of the function f on the entire curve C. Denote this restriction by $f|_C$. If the prover responds honestly with $f_{|C}$ we are in good shape, while if it responds with a wrong polynomial h, then we will show that a random point we will have $f(C(t)) \neq h(t)$ and we will be able to detect this.

We now define what we mean by a "random curve" in $\mathbb{F}^m$. A curve is simply a function $C : \mathbb{F} \to \mathbb{F}^m$. Note that this curve can be considered to be a collection of m functions $C_i : \mathbb{F} \to \mathbb{F}$, where $C(t) = \langle C_1(t), \ldots, C_m(t) \rangle$. We can now define the degree of a curve: The degree of C is simply the maximum of the degrees of the functions C_i; i.e., $\deg(C) = \max_i \deg(C_i)$.

Curves of low-degree turn out to be useful for this section, and the following proposition asserts that curves of reasonably small degree do exist passing through any small set of points. (The proof is omitted, but can be easily seen to be a consequence of the interpolation theorem for univariate polynomials.)

Proposition 1. *For any set of* (w_1) *points* $x_0, x_1, \ldots, x_w \in \mathbb{F}^m$, *there exists a unique degree* w *curve* C *with* $C(j) = x_j$ *for* $j = 0, 1, \ldots, w$.

A "random curve" through $x_1, \ldots, x_w$ is defined to the curve from the above proposition for a random value of $x_0 \in \mathbb{F}^m$. The reason we label such a curve to be random, is that most points on this curve (all except the ones that are explicitly determined) are randomly distributed (though not independently so) over $\mathbb{F}^m$. This is claimed in the next proposition.

Proposition 2. *For every $x_1, \ldots, x_w \in \mathbb{F}^m$, if $x_0 \in \mathbb{F}^m$ is picked at random and C is the unique degree w curve such that $C(j) = x_j$ for $0 \leq j \leq w$, then for any $t \notin \{1, \ldots, w\}$, $C(t)$ is a* random point *in $\mathbb{F}^m$.*

Recall that our intention is to ask a (third) prover for a description of the function $f|_C$ for some curve C. How does the prover describe this function $f|_C$? Turns out that for low degree polynomial functions, their restriction to a low-degree durve is still a low-degree polynomials. This is asserted in the next lemma.

Lemma 10. *If $P : \mathbb{F}^m \to \mathbb{F}$ is a degree d polynomial and $C : \mathbb{F} \to \mathbb{F}^m$ is a degree w curve, then $P_{|C}$ (defined by $P_{|C}(t) = P(C(t))$) is a univariate polynomial over $\mathbb{F}$ of degree wd.*

Proof. Follows by susbstituting for each variable x_i occuring in the polynomial P, the polynomial $C_i(t)$. ☐

1.2. The 3-prover MIP

We are now ready to present the promised 3-prover MIP for NP in full detail.

Input: An instance of Gap-PCS$_{1,\varepsilon}(t, m, w, s, d, q)$

Provers: There will be 3 provers Π_1, Π_2, Π_3. We will also refer to the Π_i's as proofs or oracles: the "proof" corresponding to a prover simply consists of all the responses of that prover to the various questions it might be asked. The proof Π_1 will comprise of the values of the purported "polynomial" P that is a solution to the Gap-PCS instance. Π_2 will be the "lines oracle" used to perform the low-degree test, and Π_3 will be the "curves oracle" used to perform the parallelization step.

The verifier operates as follows:

- [Random Choices:]
  1. Pick a constraint C_j of the Gap-PCS instance at random.
  2. Pick a random curve C through the w points $x_1, \ldots, x_w \in \mathbb{F}^m$ that C_j depends on. (Do this by picking a random $x_0 \in \mathbb{F}^m$ and determining the unique degree w curve C such that $C(j) = x_j$ for $j = 0, 1, \ldots, w$.)
  3. Pick a random point x on C by picking a random $t' \in \mathbb{F} - \{0, \ldots, w\}$ and setting $x = C(t')$.
  4. Pick a random line ℓ through x (i.e., pick $y \in \mathbb{F}^m$ at random and random $t'' \in \mathbb{F}$ and set $\ell = \{x + (r - t'')y : r \in \mathbb{F}\}$).
- [Queries:]
  1. Queries Π_1 for the value $P(x)$; let response be $a \in \mathbb{F}$.
  2. Queries Π_2 for the polynomial $P_{|\ell_{x,y}}$; let g be the (degree d univariate) polynomial obtained as response.
  3. Queries Π_3 for the degree wd polynomial $P_{|C}$; let h be the response.
- [Action (Accept/Reject):]
  - Reject unless $g(t'') = h(t') = a$.
  - Reject if $\langle h(1), h(2), \ldots, h(w) \rangle \in \mathbb{F}^w$ does not satisfy the constraint C_j.

– Accept otherwise.

1.3. Analysis

Presenting the analysis of the above (3-prover) MIP in full rigor with proper choice of the several parameters involved will take too long; we therefore only sketch the main ideas in the analysis. The reader is referred to [**2**] for a rigorous proof.

Completeness: It is clear that if the Gap-PCS instance is satisfiable, say by a low-degree polynomial P_0, then $\Pi_1 = P_0$ and Π_2, Π_3 defined as the restrictions of P_0 to lines and degree w curves respectively, will always satisfy the tests of the verifier. We thus have perfect completeness $c = 1$.

Soundness: Suppose we have a NO instance of Gap-PCS as input to the MIP verifier, i.e., any degree d m-variate polynomial P satisfies at most an ε fraction of the t constraints. Let $\tilde{P}$ be the response of Π_1; if $\tilde{P}$ is not δ-close to a degree d polynomial, then by the Lemma on low-degree testing from last lecture, we will have $g(t'') \neq a$ (recall that $a \stackrel{\text{def}}{=} \tilde{P}(x)$) with probability at least $\delta/2$, and thus the verifier will reject with probability at least $\delta/2$.

Now suppose $\tilde{P}$ is δ-close to a (unique) degree d polynomial P. Since we have a NO instance of Gap-PCS, with probability at least $(1-\varepsilon)$, the verifier picks a constraint C_j that is not satisfied by P. Now two cases arise:

- If $h = P_{|C}$, then $\langle h(1), \ldots, h(w) \rangle = \langle P(x_1), \ldots, P(x_m) \rangle$ and thus does not satisfy the constraint C_j, and the verifier rejects in this case.
- If $h \neq P_{|C}$, then since both $h, P_{|C}$ are degree wd polynomials, $h(t') \neq P(x)$ with probability at least $\left(1 - \frac{wd}{|\mathbb{F}|-w-1}\right)$ by the Schwartz-Zippel Lemma (since t' is a random element of $\mathbb{F} - \{0, \ldots, w\}$). Also $P, \tilde{P}$ are δ-close, so $P(x) = \tilde{P}(x)$ with probability at least $(1-\delta)$. Thus with probability at least $(1 - \delta - wd/(|\mathbb{F}| - w - 1))$, we will have $h(t') \neq a$ and the verifier will reject.

From the preceding discussion, there is a constant $\gamma > 0$, such that the verifier rejects NO instances of Gap-PCS with probability at least γ, and this gives our desired MIP characterization:

Theorem 3 ([**2**]). *There exists $\gamma > 0$ such that*

$$\text{NP} \subseteq \text{MIP}_{1,1-\gamma}\big[3, O(\log n), \text{poly}\log n\big] .$$

Part II: A Query-efficient PCP Verifier

We now turn to giving a highly query-efficient PCP verifier for NP. The verifier will only read $O(1)$ bits from the proof. On the down side, it will use polynomial randomness, and reducing the randomness to logarithmic while retaining the query complexity at $O(1)$ will be the subject of the next lecture.

2. NP $\subseteq$ PCP[poly, $O(1)$]

2.1. Quadratic Polynomials

Just as in the case of Gap-PCS, we will first show (sketch) the NP-hardness of an algebraic problem, namely "Satisfiability of quadratic polynomials" QP-SAT which tests if a set of multivariate degree two polynomials (over $\mathbb{F}_2$), say $P_1, \ldots, P_t$, have a common zero. This problem will form the basis of our new PCP verifier. We first formally define the QP-SAT problem:

QP-SAT (Satisfiability for Quadratic Polynomials)

Instance: t quadratic (degree 2) polynomials $P_1, \ldots, P_t$ on n variables $x_1, \ldots, x_n$ over $\mathbb{F}_2$.

Question: Do these polynomials have a common zero? I.e., is there an assignment $a = (a_1, \ldots, a_n)$ to $x_1, \ldots, x_n$ such that $P_j(a) = 0$ for $j = 1, 2, \ldots, t$.

Lemma 11. QP-SAT *is NP-complete.*

Proof: The problem is clearly in NP since, for YES instances, we can guess $(a_1, \ldots, a_n)$ and then verify that it is indeed a common zero. To prove NP-hardness, we reduce from CIRCUIT SAT. An instance of CIRCUIT SAT consists of a Boolean circuit C comprising of NOT gates and AND, XOR gates of fan-in two, and the goal is to decide if C has a satisfying input. It is well-known that CIRCUIT SAT is NP-complete.

To reduce CIRCUIT SAT to QP-SAT, we introduce one variable x_i for each input and for each gate of the circuit. We place a constraint for each gate of the circuit which enforces that the output of that gate is consistent with its inputs and the operation of the gate. For example, for an AND gate with associated variable x_j that receives its inputs from the gates associated with variables x_{i_1} and x_{i_2}, we would place the constraint $x_j - x_{i_1} x_{i_2} = 0$. Similar constraints are place for XOR and NOT gates. We also place a constraint corresponding to the output gate which forces it to equal 1 (so C is satisfied). Note that these constraints check for the existence of a common zero of certain degree 2 polynomial, and it is easy to see that a common zero exists for these polynomials if and only if C was satisfiable. This completes the proof. ∎

2.2. Intuition for the Verifier

Given an instance of QP-SAT the verifier must check that all there exists a such that $P_j(a) = 0$ for all $j = 1, 2, \ldots, n$. For now, pretend there were only one polynomial P (we will see how the many polynomials case reduces to this situation later). Since P is a degree two polynomial, it is of the form:

$$P(x_1, \ldots, x_n) = s_0 + \sum_{i=1}^{n} s_i x_i + \sum_{1 \le i,j \le n} c_{ij} x_i x_j \ . \tag{5}$$

where $s_0, s_1, \ldots, s_n$ and the c_{ij}'s are all elements of $\mathbb{F}_2$. We would like to check that $P(a_1, \ldots, a_n) = 0$; since we want to read *very few bits* from the proof, just asking the prover to provide $a_1, \ldots, a_n$ will not work for us. Instead we will ask the prover to write down an appropriate encoding of $a_1, \ldots, a_n$. Considering the form of P, encoding $a_1, \ldots, a_n$ using the Hadamard code and the Quadratic functions code will be useful, and we turn to the description of these codes next.

2.3. Hadamard and Quadratic Functions Code

The Hadamard Code: The Hadamard code is the most redundant linear code and consists of the evaluations of all linear functions at the message that is being encoded. More formally, given a string $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$, define $A : \mathbb{F}_2^n \to \mathbb{F}_2$ as $A(x) \stackrel{\text{def}}{=} \sum_{i=1}^n a_i x_i$. The *Hadamard encoding* of a is simply $\langle A(x) \rangle_{x \in \mathbb{F}_2^n}$. Note that a message of length n is encoded into 2^n bits under the Hadamard code. It is easy to prove that the Hadamard encodings of distinct strings differ in exactly half the bits.

Given the Hadamard encoding A of $(a_1, \ldots, a_n)$, we can compute the linear function $\sum_{i=1}^n s_i a_i$ by just one query into A, since $\sum_{i=1}^n s_i a_i = A(s)$ for $s = (s_1, \ldots, s_n) \in \mathbb{F}_2^n$. Since we are interested in evaluating some degree two polynomial P at $(a_1, \ldots, a_n)$, we will need a more redundant encoding that also includes values of quadratic functions, and thus one uses the Quadratic functions code.

The Quadratic Functions Code: Given $a_1, a_2, \ldots, a_n$, the quadratic functions code (henceforth, QF-code), encodes it by the 2^{n^2} long string $\langle Q(a) \rangle_Q$ where Q ranges over all homogeneous degree 2 polynomials over $\mathbb{F}_2$. Note that such a polynomial is specified by n^2 field elements Q_{ij}, where $Q(x) = \sum_{i,j} Q_{ij} x_i x_j$. We denote by B the QF-encoding of $a_1, \ldots, a_n$, and B defines a map $\mathbb{F}_2^{n^2} \to \mathbb{F}_2$ by $B(Q) = B(Q_{11}, \ldots, Q_{nn}) = \sum_{i,j} Q_{ij} a_i a_j$.

2.4. The "Proof"

The QP-SAT verifier will expect as proof the Hadamard and QF-encodings of a common zero $a = (a_1, \ldots, a_n)$ of the quadratic polynomials $P_1, \ldots, P_t$ in the QP-SAT instance. Note that for any degree 2 polynomial P as in Equation (5), the verifier can check $P(a) = 0$ by reading $A(s)$ and $B(c)$ from the A and B tables, thereby just making two queries. Of course, we have no guarantee that the proofs will be legal Hadamard and QF-encodings of a, and therefore as in multivariate polynomials case, we need a Testing procedure (called "Linearity Testing" in the literature) and Self-correcting procedure for the Hadamard and QF-codes.

2.5. Self-correcting the Hadamard and QF-codes

We first deal with self-correction since, as in the low-degree polynomial case, this is much easier than testing. We will present a self-correction algorithm for the Hadamard code, and the extension to the QF-code is completely straightforward. Note that Hadamard code is simply the encoding using multi-linear polynomial code, and the reader can verify that the algorithm below is in fact the same as the one for self-correcting multivariate polynomials specialized to the multi-linear case.

First let us formalize the self-correction question for the Hadamard code.

Self-Corr(A, x):

<u>Given</u>: $x \in \mathbb{F}_2^n$ and an oracle $A : \mathbb{F}_2^n \to \mathbb{F}_2$ which is δ-close to a linear function $\tilde{A}$ (for some $\delta < 1/4$ so that there is a unique δ-close linear function $\tilde{A}$ to A).

<u>Task</u>: Compute $\tilde{A}(x)$.

Lemma 12. *There is a self-correction procedure that uses $O(n)$ random bits, makes two queries and which, for* every *$x \in \mathbb{F}_2^n$, returns the correct value of $\tilde{A}(x)$ with probability at least $(1 - 2\delta)$.*

Proof: Consider the following self-correction procedure. Given $x \in \mathbb{F}_2^n$ and oracle for A which is δ-close to a linear function $\tilde{A}$, compute $\tilde{A}(x)$ as follows:

1. Pick $y \in \mathbb{F}_2^n$ at random.
2. Output $A(x + y) - A(y)$.

To prove the claim of the Lemma, note that since y and $x+y$ are random points in $\mathbb{F}_2^m$, we have $\mathbf{Pr}_y[A(y) \neq \tilde{A}(y)] \leq \delta$ and $\mathbf{Pr}_y[A(x+y) \neq \tilde{A}(x+y)] \leq \delta$. Thus with probability at least $(1-2\delta)$, we will have $A(y) = \tilde{A}(y)$ and $A(x+y) = \tilde{A}(x+y)$, and by linearity of $\tilde{A}$, this implies we output $\tilde{A}(x)$. ∎

2.6. Linearity Testing

A function $f : \mathbb{F}_2^m \to \mathbb{F}_2$ is called *linear* if $f(x+y) = f(x) + f(y)$ for all $x, y \in \mathbb{F}_2^m$. This is equivalent to $< f(x) >_{x \in \mathbb{F}_2^m}$ being a Hadamard codeword. The verifier for QP-SAT we wish to construct, needs to check linearity of both the A and B tables it is presented as proof, and thus Linearity Testing is a crucial component in this construction. It is also a very natural combinatorial problem in its own right.

Formally, the specification of the linearity testing problem is the following:

Given: $\delta > 0$; oracle $f : \mathbb{F}_2^m \to \mathbb{F}_2$.

Task: Test if f is δ-close to a linear function $\tilde{f}$.

The following asserts the existence of a good Linearity test:

Lemma 13. *There is a Linearity Test which uses $O(m)$ random bits, makes just 3 queries into f, and has the following properties:*

(i) *It accepts with probability* 1 *if f is linear.*
(ii) *It accepts with probability at most $(1-\delta)$ if f is not δ-close to linear.*

Proof: The test itself is quite simple:

1. Pick $x, y \in \mathbb{F}_2^m$ at random
2. Accept iff $f(x) = f(x+y) - f(y)$.

It is clear that the test makes only 3 queries into f and that it always accepts if f is a linear function. The soundness claim (ii) above is, however, not straightforward to prove, and was first proved (with a weaker dependence of the acceptance probability on the closeness to linearity) by Blum, Luby and Rubinfeld [**13**] in their seminal paper. The result in the form claimed was shown by Bellare, Coppersmith, Håstad, Kiwi and Sudan [**8**]. ∎

2.7. Testing "Consistency"

From the preceding two subsections, we are equipped to test that the tables A, B which are purportedly the Hadamard and QF-encodings of some $(a_1, \ldots, a_n)$ (which ought to be a common zero of the QP-SAT instance we are testing for satisfiability) are close to linear functions and to self-correct them. Now, suppose we have linear functions $\tilde{A} : \mathbb{F}_2^n \to \mathbb{F}_2$ and $\tilde{B} : \mathbb{F}_2^{n^2} \to \mathbb{F}_2$ that are δ-close to A and B respectively. Since $\tilde{A}$ is linear, there exists $a = (a_1, \ldots, a_n)$ such that $\tilde{A} = \chi_a$, i.e. $\tilde{A}(x) = \sum_{i=1}^n a_i x_i$ for all $x \in \mathbb{F}_2^n$. Similarly there exists $b = (b_{11}, \ldots, b_{nn})$ such that $\tilde{B} = \chi_b$, i.e. $\tilde{B}(q) = \sum_{i,j} b_{ij} q_{ij}$ for all $q \in \mathbb{F}_2^{n^2}$. But we would like $\tilde{B}$ to be the QF-encoding of a, and thus we need to check "consistency", namely that $b_{ij} = a_i a_j$ for all $1 \leq i, j \leq n$.

Lemma 14. *Given oracle access to $A : \mathbb{F}_2^n \to \mathbb{F}$ and $B : \mathbb{F}_2^{n^2} \to \mathbb{F}_2$ which are δ-close to χ_a, χ_b respectively for some $a \in \mathbb{F}_2^n$ and $b \in F_2^{n^2}$, there is a probabilistic test that uses $O(n^2)$ random bits, makes* 6 *queries and satisfies the following properties:*

(i) *If $A = \chi_a$, $B = \chi_b$ and $b_{ij} = a_i a_j$ for all i, j, then the test always accepts.*

(ii) *If there exist i, j such that $b_{ij} \neq a_i a_j$, then the test rejects with probability at least $(\frac{1}{4} - 6\delta)$.*

Proof: The test does the following:

1. Pick $x, y \in \mathbb{F}_2^n$ at random; Let $q \in \mathbb{F}_2^{n^2}$ such that $q_{ij} = x_i y_j$ for $1 \leq i, j \leq n$.
2. Accept iff $\mathsf{Self\text{-}Corr}(A, x) \cdot \mathsf{Self\text{-}Corr}(A, y) = \mathsf{Self\text{-}Corr}(B, q)$.

(In the above $\mathsf{Self\text{-}Corr}(A, x)$ stands for the element returned by calling the self-correction procedure from Lemma 12.) Clearly the above only uses $O(n^2)$ random bits. Since the self-correction procedure makes 2 queries, the above test makes a total of 6 queries. Also the completeness condition (i) is clearly met.

Now consider the soundness Case (ii). Define $n \times n$ matrices M_1, M_2 over $\mathbb{F}_2$ as: $\{M_1\}_{ij} = a_i a_j$ and $\{M_2\}_{ij} = b_{ij}$. By hypothesis, there exist i, j such that $a_i a_j \neq b_{ij}$, so we have $M_1 \neq M_2$. Since A is δ-close to χ_a and B is δ-close to χ_b, by Lemma 12, with probability $(1 - 6\delta)$, the test in Step (2) above checks that $\chi_a(x) \cdot \chi_a(y) = \chi_b(q)$, or in other words $\sum_{i,j} a_i a_j x_i y_j = \sum_{i,j} b_{ij} x_i y_j$ which is the same as $x^T M_1 y = x^T M_2 y$. Since $M_1 \neq M_2$, this happens with probability at most $3/4$ for a random choice of $x, y \in \mathbb{F}_2^n$ (this is easy to show). The overall probability of acceptance is thus at most $3/4 + 6\delta$, as claimed. ∎

2.8. Putting Everything Together

To give the verifier in the final form, we need one more trick. To verify satisfiability of the QP-SAT instance, we need to check $P_j(a)$ for every $j = 1, 2, \ldots, t$. For efficient checking, we need to "aggregate" these into a single constraint. This is done as follows:

1. Pick $r = (r_1, \ldots, r_t) \in \mathbb{F}_2^t$ at random.
2. Replace the constraints $P_j(a) = 0$ for all $j = 1, \ldots, t$ by the single constraint $P_r(a) = 0$ where

$$P_r \stackrel{\text{def}}{=} \sum_{j=1}^{t} r_j P_j \ . \tag{6}$$

The key fact about P_r is captured by the following easy lemma.

Lemma 15. (i) *If $P_j(a) = 0$ for all j, then $P_r(a) = 0$.*
(ii) *If there exists j such that $P_j(a) \neq 0$, then $P_r(a) \neq 0$ with probability (exactly) $1/2$.*

The Verifier: We (finally!) present the verifier with all components put together:

Input: An instance $(n, P_1, \ldots, P_t)$ of QP-SAT.

Goal: Verify that the polynomials P_j have a common zero $a \in \mathbb{F}_2^n$.

Expected Proof: Tables $A : \mathbb{F}_2^n \to \mathbb{F}_2$ and $B : \mathbb{F}_2^{n^2} \to \mathbb{F}_2$ which are supposedly the Hadamard and QF encodings of a common zero $a \in \mathbb{F}_2^n$ of the P_j's.

The verification procedure operates as follows:

1. Perform a Linearity Test on A, B (Lemma 13). Reject it the test fails.
2. Perform the "Consistency check" (Lemma 14) on A, B. Reject if the check fails.
(We have now verified with good confidence that A, B are δ-close to χ_a, χ_b respectively where $b_{ij} = a_i a_j$ for all i, j.)

3. Pick $r \in \mathbb{F}_2^t$ at random and compute the (coefficients of the) polynomial $P_r = \sum_j r_j P_j$. Let
$$P_r(x_1, \ldots, x_n) = s_0 + \sum_{i=1}^{n} s_i x_i + \sum_{1 \le i,j \le n} c_{ij} x_i x_j \ .$$
Let $s = (s_1, \ldots, s_n)$ and $c = (c_{11}, \ldots, c_{nn})$.
4. Accept iff $s_0 + \mathsf{Self\text{-}Corr}(A, s) + \mathsf{Self\text{-}Corr}(B, c) = 0$. (This corresponds to checking that $P_r(a) = 0$.)

Note that the above verifier used $O(t + n^2) = O(n^2)$ random bits (from the proof of Lemma 11, we can assume $t \le n^2$ – in fact $t = O(n)$ – for the hard instance of QP-SAT). The verifier also makes only 16 queries in all (6 in Step 1, 6 in Step 2, and 4 in Step 4 above). From the NP-hardness of QP-SAT (Lemma 11) and Lemmas 15, 13, 12 and 14, we can show that the verifier has completeness 1 and soundness at most $(1 - \varepsilon)$ for some $\varepsilon > 0$ (we leave it to the reader to fill in the details, or see [**2**]). We thus get:

Theorem 4 ([**2**]). *There exists $\varepsilon > 0$ such that*
$$\mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\varepsilon}\big[O(n^2), 16\big] \ .$$

LECTURE 4
Proof Composition and the PCP Theorem

1. Where are we?

Recall from the last lecture that we now have the following two proof systems for NP. The first is a 3-prover MIP for NP whose verifier uses $O(\log n)$ randomness, receives answers of poly $\log n$ bits from each of the 3 provers, and decides to accept or reject based on the verdict of a circuit of size poly $\log n$ on the (concatenation of the) received answers. The second is a PCP for NP whose verifier makes only 16 queries into the proof and uses $O(n^2)$ randomness. From the high level, the former proof system has small randomness, but large query complexity; while the latter has small query complexity, but large randomness. In contrast, our goal is to have small randomness and small query complexity, and it seems neither the PCPs obtained so far give us what we want. In this lecture we describe a method of composing proofs together that magically puts the two PCPs together to get (close) to our goal. Specifically composition takes an "outer PCP" with small randomness and an "inner PCP" with small query complexity and combines them to get a "composed PCP" with small randomness and small query complexity. Composition also maintains some basic properties on completeness and soundness, and in particular it preserves perfect completeness and the property of soundness being bounded away from 1.

In this lecture, we first illustrate composition with an example. This example already builds a PCP with much better parameters than we know of. But composition can take us further. We describe from a high-level how composition applies to a fairly general class of PCPs, and assert that the PCPs we have seen so far are amenable to composition. Modulo this assertion, we then obtain a proof of the PCP theorem. In fact, the composition theorem even takes us further — to the optimal PCP theorem, and we list some of the steps that yield this stronger conclusion.

2. Composing the Verifiers

2.1. A first attempt

Composition is based on the following simple observation. Suppose we have a power PCP (call it the inner verifier) that knows how to verify that circuits are satisfiable. Maybe we can use this PCP to make the verification step of another

PCP (called the outer verifier) easier. Note that the outer verifier typically behaves as follows: It tosses some random coins and based on these it devises a strategy on how to check the proof. In particular it generates some queries, and then prepares a Boolean predicate that will determine if a collection of responses to the queries are acceptable or not. Typically this Boolean predicate is described by a small circuit C. The outer verifier then sends the queries to some provers, and then obtains responses to these queries. It then plugs in these responses into the circuit C, to determine whether to accept or not. The composition paradigm is motivated by the intuition that it should be possible to use the inner verifier to verify that C is satisfied by these responses. However the exact description of this paradigm involves some surprisingly subtle issues and we motivate this by describing an attempt to compose the two PCP verifiers of the previous lecture together.

1. Start with the verification procedure of the 3-prover MIP.
2. Prepare queries q_1, q_2, q_3 and a small circuit C that determines the accept/reject decision of the verifier.
3. Send the queries to the provers, but now instead of just receiving the responses a_1, a_2, a_3 from the three provers (which would cause the query complexity to be poly $\log n$), ask the prover to write down a proof that (a_1, a_2, a_3) is a satisfying assignment to the circuit C using the encoding standard of the 16 query PCP verifier. (Here we are using the fact that Circuit Sat is in NP and thus there exists a PCP for the fact that a_1, a_2, a_3 satisfies C.)

Note that above applies a PCP recursively to the task of checking that a_1, a_2, a_3 is a satisfying assignment to C, and thus the above is also referred to in the literature as "recursive proof checking" or "recursive composition of proofs". The idea of proof composition originated in the work of Arora and Safra [**3**] and has been a crucial component in all PCP constructions that followed.

Analyzing the above Composition: The above composed verifier makes only 16 queries and uses $O(\log n)$ randomness for the initial verification process in Steps 1 and 2 (called "outer" verification) and another $O((\text{poly}\log n)^2) = \text{poly}\log n$ randomness when it simulates the second verifier in Step 3 (called "inner" verification), for a total of poly $\log n$ randomness. Thus it at least has better quantitative parameters than both of the verifiers we started with! The verifier, however, does not inherit the soundness of the two original verifiers. The reason is that we are asking the prover for a proof that there exists an input (a_1, a_2, a_3) that satisfies C, which is not the same as asking the prover to prove that a given triple a_1, a_2, and a_3 combine together to satisfy C. In particular, when the query q_1 is asked in a different context, we do not check to verify that the answer to q_1 in the other context is the same as the answer in the current context. Thus the prover can "cheat" by using a satisfying assignment for C that has nothing to do with the 3 answers that would have been given by the MIP prover. (To consider a simple but illustrative example, consider a single prover verifier for 3SAT, who just picks a random clause in a given formula, whose satisfiability is to be verified, and then asks a prover for the value of the literals in the clause. Clearly the prover would have no problem convincing the verifier that this clause can be satisfied, and so the verifier accepts with probability 1, independent of the satisfiability of the formula. The composition method described above is functioning analogous to this verifier and hence does not have a hope to testing anything.)

To fix the bug above, somehow we need to make sure that the various answers given by the prover for the various tests are all "consistent" (i.e. the different clauses referring to the same variable use the same assignment to that variable) and would hence "glue together" to give a *single global assignment* that satisfies all (or most of) the clauses.

In a nutshell, we need to ensure *consistency* between the answers committed to by the prover in response to various different circuit tests C, so that we can argue that if the composed verifier accepts with large probability then one can fix responses for the three provers in the "outer" MIP that will cause the MIP verifier to accept with large probability. Together with the soundness of the MIP, this will imply the soundness of the composed verifier.

2.2. A modified composition scheme

We now discuss at an intuitive level the way to fix this problem in the composed verifier. The idea is to force the prover to commit to responses to individual queries (e.g. q_1) by writing down an appropriate (e.g. the Hadamard) encoding of the answers. We will view such an encoding as a table (denoted Π_{q_1}) that we wish to probe minimally, but something that already commits to the answer to query q_1. In addition to providing such a table for every query that the 3-prover MIP can possible ask, the prover for the composed verifier is also asked to write down proofs Π that (a_1, a_2, a_3) satisfies C (for various choices of q_1, q_2, q_3, C made by the MIP verifier in the first stage of the composed verification). The verifier will now check that $C(a_1, a_2, a_3)$ accepts by making queries to the corresponding proof Π of the inner (16-query) PCP, and in addition will perform consistency checks between the various components of Π and the proofs $\Pi_{q_1}, \Pi_{q_2}, \Pi_{q_3}$. More specifically, for the verifiers we have, we can require Π_{q_1} to be the Hadamard encoding A_1 of the response a_1, and recall from the last lecture that the proof Π for the "inner" PCP includes the Hadamard encoding, say B, of $a_1 \circ a_2 \circ a_3$ (here $\circ$ denotes the concatenation operation). The consistency check between Π and Π_{q_1} will now check that $A_1(x) = \mathsf{Self\text{-}Corr}(B(x \circ 0^b))$ for a random x of length $|a_1|$ (here b is the suitable number of zeroes padded at the end of x). Note that the query complexity of this composed verifier will be 16 plus the 3 queries made in each of the three consistency checks, for a total of 25 queries.

We have been very informal in our description of proof composition, and the interested reader can find the formal details in [**3, 2**]. We now give a semi-formal summary of the composed verifier for easy reference.

Composed PCP verifier for NP**:**

Structure of expected proof: The verifier has oracle access to a proof Π which is expected to have the encodings of all the answers of the 3 provers of the MIP (as per some suitable error-correcting code) for the various possible queries of the MIP verifier. More specifically, for $1 \le i \le 3$ and query q_i of the MIP verifier to prover i, $\Pi(i, q_i, \cdot)$ is the encoding of the response a_i of prover i to query q_i. In addition, for each random choice R of the MIP verifier, $\Pi(0, R, \cdot)$ will be the encoded proof (for the inner PCP system) of the satisfiability of the circuit C_R corresponding to R computed by the MIP verifier.

Given access to the oracle Π, the verifier operates as follows:

- Pick random string R as per the 3-prover MIP verifier (from last lecture) and generate queries q_1, q_2, q_3 to the three provers and a circuit C.
- Let $A_i(\cdot) = \Pi(i, q_i, \cdot)$ and $B(\cdot) = \Pi(0, R, \cdot)$.
- Now perform "inner verification" for (the satisfiability of) C with oracles A_1, A_2, A_3, B as below:
  - Run the 16 query PCP verifier (from last lecture) on oracle B with input C (we are testing that B encodes a satisfying assignment to C).
  - Perform consistency checks on oracle pairs (A_1, B), (A_2, B) and (A_3, B).

One can formalize the discussion of the preceding sections and prove that the above verifier (which we already argued uses $\text{poly}\log n$ randomness and makes only $O(1)$ queries – in fact it makes only 25 queries) also has soundness bounded away from 1, and this gives us:

Theorem 5. *There exists a $\gamma > 0$ such that* $\text{NP} \subseteq \text{PCP}_{1,1-\gamma}[\text{poly}\log n, 25]$.

Composition as a paradigm: The basic ingredients of composition abstracted from the preceding construction are the outer and inner verifiers. The outer verifier is an MIP verifier with a small number of provers p and whose acceptance predicate is computed by a small circuit, and which has very low soundness error. The answer size of the MIP governs the size of the problem passed on to the inner verifier.

The inner verifier has low query complexity q and must be able to verify the commitment to a proof rather than the mere existence of one. The composed verifier starts out by simulating the outer verifier and after the outer verifier picks a circuit C which computes its acceptance predicate, the composed verifier uses the inner verifier on input C. If suitable conditions are met, then one can compose the outer and inner verifier to get a verifier that combines the randomness efficiency of the outer verifier with the query efficiency of the inner verifier.

Formalism of the notion of outer and inner verifiers and exactly how they compose together can be found in work of Arora and Safra [**3**]. Several refinements to their "Composition Theorem" can be found in several later works like [**2, 9**].

3. The PCP Theorem

To prove the PCP Theorem we need to reduce the randomness of the verifier from Theorem 5 to logarithmic from poly-logarithmic. The reason we had $\text{poly}\log n$ randomness was that the outer MIP in the above composition had $\text{poly}\log n$ answer and circuit size and the inner verifier used a quadratic number of random bits (as a function of *its* input length). Thus in order to reduce the overall randomness, we would like to reduce the answer size of the outer MIP.

It turns out that the 3-prover MIP construction from the last lecture also yields an inner verifier which can be used to show that $\forall \varepsilon > 0$, $\exists \delta > 0$ such that

$$\text{MIP}_{1,1-\varepsilon}[p, r, a] \subseteq \text{MIP}_{1,1-\delta}[p+3, r + O(\log a), \text{poly}\log a] \ .$$

(Such a result is shown in [**2**].) Combining with the MIP characterization $\text{NP} = \text{MIP}[3, O(\log n), \text{poly}\log n]$ from the previous lecture, this gives, upon composing the MIP verifier with itself as the inner verifier (it is shown in [**2**] how to modify this verifier to also function as an inner verifier), the characterization $\text{NP} = \text{MIP}[6, O(\log n), \text{poly}\log\log n]$. Composing this 6-prover MIP verifier with the $O(1)$-bit, quadratic randomness verifier from [**2**] which was discussed in the last lecture, gives a logarithmic randomness, $O(1)$ query complexity verifier for NP with perfect completeness and soundness bounded away from 1, or in other words the PCP Theorem! We would thus get:

Theorem 6 ([**3, 2**]). *There exists an* $\varepsilon > 0$ *such that*

$$\mathrm{NP} = \mathrm{PCP}_{1,1-\varepsilon}[O(\log n), 34] \ .$$

4. Towards Optimal PCPs

There are a number of respects in which one can hope to improve Theorem 6. This has been the focus of a large body of works including [**15, 11, 9, 20, 21, 19, 28, 27, 26**]. One specific question, for example, is: What is the minimum number of queries required to obtain a desired soundness error? The quest for better (and optimal) PCP constructions has also been motivated by applications to hardness of approximations where improvements in the underlying PCPs often translate directly into improvements in the related inapproximability result that it gives.

We will only give an overview of what is involved in obtaining optimal PCPs and not give any technical details or prove any of the claims. There are two main ingredients in obtaining optimal PCP constructions. The first one is improved constructions of MIPs, specifically those with very few provers, preferably 2 provers, with extremely low soundness error and at the same time having small answer sizes and logarithmic randomness. The second ingredient(s) are "optimal" inner verifiers that are tuned to simplifying verifiers for 2-prover proof systems.

We will now elaborate a little on constructions of 2-prover proof systems. The starting point for such a construction is the PCP theorem (Theorem 6) itself: $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\varepsilon}[O(\log n), 34]$. One can convert such a PCP verifier into a verifier for a 2-prover proof system using a technique in [**16**] as follows:

- Pick a random string R and generate queries $q_1, \dots, q_{34}$ (as the PCP verifier would do). Send *all* queries to Prover 1.
- Pick a random index $i \in \{1, \dots, 34\}$ and send query q_i to Prover 2.
- Accept iff answers of Prover 1 make the PCP verifier accept, and the answer of Prover 1 on query q_i is *consistent* with the response of Prover 2.

It is clear that the above verifier has logarithmic randomness and receives $O(1)$ size answers. It also clearly has perfect completeness since the original PCP had perfect completeness. It is not difficult to show that the soundness is bounded away from 1, and thus this gives us a MIP with 2-provers as a starting point. But the soundness is very close to 1 and we would like to improve the soundness while keeping the answer size and randomness small.

The natural approach to reducing the error is repeating the verifier's action several times with independent random tosses, but doing this sequentially would increase the number of rounds of interaction between the verifier and the provers. The approach instead is to repeat the verification many times *in parallel* (with independent coin tosses), but, unlike the sequential repetition case, it is now no longer obvious that the soundness error goes down exponentially with the number of repetitions.

An important result of Raz [**24**], called the *Parallel Repetition Theorem* shows that this is indeed the case (the result holds for all 2-prover systems where the verifier is "canonical" in the sense that its acceptance condition is a check that a certain projection of the answer of Prover 1 equals the answer of Prover 2). The proof of this result is complicated, but for our purposes it suffices to understand that it implies the following error reduction fact for MIPs: For every $\varepsilon > 0$ and integer a, there exists an $\varepsilon' > 0$ such that for all $k \geq 2$, a canonical verifier for a 2-prover MIP with randomness r and which receives answers of size a and 1 from the two provers and has soundness $(1-\varepsilon)$, can be converted into one with answer size at most ka, randomness at most kr, and soundness error $(1-\varepsilon')^k$. Informally, the transformation is

$$\mathrm{MIP}_{1,1-\varepsilon}[2, r, a] \longrightarrow \mathrm{MIP}_{1,(1-\varepsilon')^k}[2, kr, ka] \ .$$

The above enables us to construct 2-prover MIPs for NP with very low soundness error and constant answer sizes. We do not elaborate on the inner verifiers, but to obtain improved PCPs one takes such a 2-prover MIP and composes it with a suitable inner verifier. For the optimal constructions, it turns out that one uses inner verifiers which take the encoding of the answers of the 2 provers of the outer MIP by a code called the *Long Code* (first defined in [**9**]) and then verify , using extremely query-efficient procedures, that these are indeed “close to” encodings of valid answers that would make the verifier of the outer MIP accept. It turns out that using some machinery from Discrete Fourier Analysis, such Long Code based inner verifiers can often be analyzed optimally, and this approach was pioneered by Håstad in a series of striking results [**20, 21**]. We do not elaborate on this further, but just mention that one such tight result from [**20**] is the following, which shows that just 3 queries are enough to obtain a soundness error close to 1/2 (it is known that one cannot do better [**29**]).

Theorem 7 ([**20**]). *For any $\varepsilon > 0$, we have* $\mathrm{NP} = \mathrm{PCP}_{1-\varepsilon,1/2}[O(\log n), 3]$.

5. Roadmap to the Optimal PCP

Before winding up, we give a quick high-level recap of the road to a complete proof of the optimal PCP construction from Theorem 7 above. The main steps are the following:

1. 3-prover MIP verifier for NP ($\mathrm{NP} = \mathrm{MIP}_{1,1-\gamma}[3, O(\log n), \mathrm{poly}\log n]$) [**2**]
2. Compose the above verifier with itself (using the paradigm of composition from [**3**]) to get $\mathrm{NP} = \mathrm{MIP}_{1,1-\gamma'}[6, O(\log n), \mathrm{poly}\log\log n]$ [**2**].
3. An $O(1)$ query, $O(n^2)$ randomness verifier for NP from [**2**] ($\mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\varepsilon}[O(n^2), O(1)]$).
4. Compose the verifier from Step 2 with the verifier from the previous step to get $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\varepsilon'}[O(\log n), O(1)]$. At this stage we have the PCP Theorem [**3, 2**].
5. Obtain a 2-prover MIP for NP from the above PCP verifier (as in [**16**]) and then apply Raz's Parallel Repetition Theorem [**24**] to prove that for all $\delta > 0$, $\mathrm{NP} \subseteq \mathrm{MIP}_{1,\delta}[2, c_\delta \log n, a_\delta]$ where c_δ and a_δ are constants depending only on δ.
6. Compose the verifier from above 2-prover proof system with a 3-query inner verifier from [**20**] to get (one) optimal PCP Theorem: $\mathrm{NP} = \mathrm{PCP}_{1-\varepsilon,1/2}[O(\log n), 3]$ for every $\varepsilon > 0$.

Note that the main omissions from the above path in our discussion has been the Parallel Repetition Theorem and a description and analysis of Håstad's optimal inner verifier.

The proof of the PCP Theorem is thus quite complicated and puts together several ingredients. It is an important open question whether any portions (or all) of the proof can be simplified. A good starting point in approaching this question would be to first look at simpler constructions of what are called *locally checkable codes*. These are codes with polynomially small rate such that given a string one can determine if it is a codeword or is sufficiently far off from any codeword by just looking at the symbols in $O(1)$ positions of the string. Such codes are implied by the PCP Theorem and the only construction we know of such codes goes via the PCP Theorem. An alternative, simpler construction of such codes might enable a shot at simpler proofs of the PCP Theorem, and would also be extremely interesting and important in its own right.

BIBLIOGRAPHY

1. Sanjeev Arora and Carsten Lund. Hardness of approximations. In *Approximation Algorithms for NP-hard Problems,* D. Hochbaum (Ed.), PWS Publishing, 1996.
2. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in *Proceedings of FOCS'92.*
3. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in *Proceedings of FOCS'92.*
4. László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 421-429, Providence, Rhode Island, 6-8 May 1985.
5. László Babai, Lance Fortnow, Leonid Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 21-31, New Orleans, Louisiana, 6-8 May 1991.
6. László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. Preliminary version in *Proceedings of FOCS'90.*
7. Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. *Proc. of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS Vol. 415, Springer-Verlag, 1990.
8. Mihir Bellare, Don Coppersmith, Johan Håstad, Marcos Kiwi and Madhu Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6), pp. 1781-1795, 1996.
9. Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCP's and non-approximability – towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998. Preliminary version in *Proceedings of FOCS'95.*
10. Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alexander Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 294-304, San Diego, California, 16-18 May 1993.
11. Mihir Bellare and Madhu Sudan. Improved non-approximability results. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of*

Computing, pages 184-193, Montreal, Quebec, Canada, 23-25 May 1994.

12. Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 113-131, Chicago, Illinois, 2-4 May 1988.
13. Manuel Blum, Michael Luby and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.
14. Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in *Proceedings of FOCS'91*.
15. Uriel Feige and Joe Kilian. Two prover protocols – low error at affordable rates (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 172-183, Montreal, Quebec, Canada, 23-25 May 1994.
16. Lance Fortnow, John Rompel, and Michael Sipser. On the power of multiprover interactive protocols. *Theoretical Computer Science*, 134:545–557, 1994.
17. Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115-1145, November 1995.
18. Shafi Goldwasser, Silvio Micali and Charles Rackoff. The knowledge complexity of interactive proofs. *SIAM Journal on Computing*, 18:186–208, 1989.
19. Venkatesan Guruswami, Daniel Lewin, Madhu Sudan and Luca Trevisan. A tight characterization of NP with 3-query PCPs. *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998.
20. Johan Håstad. Some optimal inapproximability results. Technical Report TR97-037, Electronic Colloquium on Computational Complexity, 1997. Preliminary version in *Proceedings of STOC'97*.
21. Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *ECCC Technical Report TR97-038*. (Preliminary versions in *Proceedings of FOCS '96 and STOC'96*).
22. Howard Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *38th Annual Symposium on Foundations of Computer Science*, pages 406-415, Miami Beach, Florida, 20-22 October 1997.
23. Alexander Polishchuk and Daniel Spielman. Nearly-linear size holographic proofs. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 194-203, Montral, Qubec, Canada, 23-25 May 1994.
24. Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. Preliminary version in *Proceedings of STOC'95*.
25. Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.
26. Alex Samorodnitsky and Luca Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 191–199, Portland, Oregon, 21-23 May, 2000.

    191–199.

27. Madhu Sudan and Luca Trevisan. Probabilistically checkable proofs with low amortized query complexity. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 18–27, Palo Alto, California, 8-11 November, 1998.
28. Luca Trevisan. Recycling queries in PCPs and in linearity tests. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 299–308, Dallas, Texas, 23-26 May, 1998.
29. Uri Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.