SIGACT Complexity Column: PCPs with Small Soundness Error

Irit Dinur

Abstract

The soundness error of a PCP verifier is the probability (usually denoted ε) that the verifier accepts an incorrect input. We are interested in the smallest possible values of ε for which the PCP theorem holds, and in particular whether the theorem holds when ε is an inverse polynomial function of the input length. We discuss the 'sliding scale conjecture' of [BGLR93, LY94] and related questions. We then sketch some of the existing approaches and constructions of PCPs with sub-constant soundness error.

1 Introduction

The PCP theorem [AS98, ALM$^+$98] gives a format for writing NP proofs that enables probabilistic checking. Formally, a *restricted verifier* is a randomized algorithm that reads an input x of length n, and then uses $r(n)$ random bits and makes $q(n)$ queries to a proof, and decides if to accept or reject the input. A language L is said to be accepted by such a verifier if

- If $x \in L$ there is a proof that causes the verifier to accept with probability one.
- If $x \notin L$ then no matter what the proof is, the verifier accepts with probability at most ε.

The class of all languages accepted by such a verifier is denoted by $PCP_{1,\varepsilon}[r(n), q(n)]$ and the famous PCP theorem is

Theorem 1 (The PCP Theorem, [AS98, ALM$^+$98]) *There exist constants $\varepsilon < 1$ and $q > 1$ such that*

$$NP \subseteq PCP_{1,\varepsilon}[O(\log n), q].$$

The parameter q is naturally called the query complexity, and ε is called the *soundness error* since it bounds the probability that the verifier erroneously accepts an input that is supposed to be rejected. Such a system is said to have *perfect completeness* since there is zero probability of rejecting $x \in L$ when supplied with a correct proof.

For $q = O(1)$ and a constant $\varepsilon < 1$ a popular interpretation of the theorem is that it gives a robust format for writing NP witnesses: the verifier will accept a false statement with probability bounded away from one. This probability can be further reduced to ε^t by running t independent copies of the verifier and accepting if and only if they all accept. If one is willing to ignore constants, this means that any NP language has a witness format that can be verified with error that is a constant arbitrarily close to 0, by reading only a constant number of proof-bits. Of course, in terms of parameters this repetition will increase the number of queries from q to $q \cdot t$.

1.1 Trading the number of queried bits for soundness error

What is the "correct" tradeoff between the number of bits read from the proof and the soundness error? Clearly if the verifier only looks at t bits from the proof, the soundness error will be at least 2^{-t}. The reason is that even a random proof will cause the verifier to accept with probability 2^{-t}, regardless of the correctness of the statement. This lower bound is nearly matched by the soundness error of $2^{-\Omega(t)}$ guaranteed by the PCP theorem. This tradeoff holds for all values of t, including when t is a function of the input size n, by simple sequential amplification, described next.

Theorem 2 *For every* $t : \mathbf{N} \to \mathbf{N}$

$$NP \subseteq PCP_{1,\exp(-\Omega(t(n)))}[\,O(\log n + t(n))\,,\; t(n)\,].$$

Note that for $t(n) > n$ the theorem is trivial since the verifier can simply read the entire NP witness in that case and have no error at all. For $t(n) \leq O(\log n)$ the verifier uses at most a logarithmic amount of randomness, which means that the proof still has polynomial size.

Proof Sketch: Let V be the verifier whose existence is asserted by Theorem 1. We run this verifier $t(n)/q$ times (we divide by $q = O(1)$ only to ensure that our new verifier reads at most $t(n)$ proof bits), and accept iff all runs resulted in acceptance. Doing this in a randomness-efficient manner (by recycling random bits using e.g. expander walks) the soundness error becomes $\exp(-\Omega(t(n)))$ using only $O(\log n + t(n))$ random bits. ■

1.2 Many proof bits, few proof queries

So far we have encountered two out of the three parameters that are to be the focus of our discussion: the soundness error (ε) and the number of queries (q). The third parameter is called the *alphabet size* and is usually denoted by Σ. It refines the q parameter by allowing the answer size of each query to be more than one bit. Thus we make a distinction between the number of queries and the number of bits in each answer (this equals $\log|\Sigma|$).

Think of the generalization of the verification process to the case where proofs are written over larger alphabets, and we restrict the verifier to read at most q symbols from the proof as before. Alternatively, one may think of the proof as being partitioned into small "pieces", and the verifier is allowed to read only a certain number of pieces. We add the subscript Σ and denote by $PCP_{1,\varepsilon(n)}[r(n), q(n)]_{\Sigma(n)}$ the class of all languages accepted by a verifier that uses $r(n)$ random bits, reads $q(n)$ symbols from a proof written over an alphabet Σ and has soundness error ε.

How do these two models compare? A verifier that reads at most q pieces each consisting of at most $w = \log|\Sigma|$ bits, can always be simulated by a verifier that reads qw bits,

Claim 3 $PCP_{1,\varepsilon}[r, q]_\Sigma \subseteq PCP_{1,\varepsilon}[r, q\log|\Sigma|]_{\{0,1\}}$.

Is this containment an equality? this is only known to hold for a partial range of the parameters. Of particular interest is the following question:

Is it possible to convert a verifier that reads t proof bits into a verifier that reads $O(1)$ pieces each consisting of $O(t)$ bits, while maintaining the same soundness error?
Equivalently,

$$PCP_{1,\varepsilon}[O(\log n), t(n)]_{\{0,1\}} \overset{?}{\subseteq} PCP_{1,\varepsilon}[O(\log n), O(1)]_{\{0,1\}^{t(n)}}$$

This question is called *parallelization* because such a reduction can be viewed as "parallelizing" the t queries into $O(1)$ (and ultimately 2). A naive approach for solving this problem is to bunch together t bits from the original proof into a single (large alphabet) symbol, and then simulate the original verifier over the new proof. Of course a single bit in the original proof will potentially participate in several "bunches". The new proof will only be sound if we somehow manage to guarantee that the bit is assigned the same value in every occurrence. This is the notorious problem of *consistency.*

We note that the number of queries can always be reduced from t to two if one only cares for a soundness error on the order of $1 - O(1/t)$. In such a case the naive parallelization of bunching together t symbols into one, and performing a simple comparison consistency check will work[1]. However, even for small t this soundness error does not approach zero. So for small values of the soundness error, the parallelization question becomes interesting. We list several motivations for studying this question,

- *Hardness of Approximation.* There is a fundamental connection between PCPs and hardness of approximation. This connection, discovered by [FGL$^+$96], served as motivation for much of the consequent development in PCPs. Reductions for proving hardness of approximation take a PCP system to an approximation problem. Often the soundness error translates directly to the factor for which hardness of approximation is proven. However, the seemingly optimal Theorem 2 usually fails to give any improvement over the basic PCP theorem because of the unbounded number of queries. One may hope to get stronger inapproximability results for certain approximation problems by making progress on this question. An example that comes to mind is polynomial hardness for lattice problems, although we are not aware of a direct reduction.

- *Composition.* An essential ingredient in PCP constructions is the powerful composition-recursion technique of [AS98]. The idea is to have the verifier call an *inner verifier* to do part of the work, supplying the inner verifier with an additional proof. A key issue in such a scheme is that of consistency between the various inner proofs, and the only way we know how to solve it is by controlling the number of queries. Therefore, better "parallelization" techniques may result in better PCP constructions.

- *Inherent interest.* One may argue that a proof written over non-binary alphabets is natural enough an object to merit study in its own right.

Tying together the first and third motivation item is the fact that the parallelization question is *equivalent* to a question about the hardness of approximating certain constraint satisfaction problems.

1.3 Hardness of Constraint Satisfaction Problems (CSPs)

One way to see the [FGL$^+$96] connection (tying PCPs and inapproximability) is by reformulating the PCP theorem in terms of a hardness of approximation result for a general constraint satisfaction problem.

An instance of q-CSP(Σ) (parameterized by an integer $q \in \mathbf{N}$ and a finite set Σ) is a set $\Pi = \{\pi_1, \ldots, \pi_m\}$ of variables that take values over some alphabet Σ, together with a set of

[1]More concretely: the new verifier simulates the old verifier by reading the appropriate t-tuple of bits. It then also selects a random bit in the t-tuple and checks that it is consistent with a random occurrence of this bit in another t-tuple.

q-ary constraints. Formally, a q-ary constraint specifies a q-tuple of indices $(i_1, \ldots, i_q) \in [m]^q$ and a predicate $\varphi : \Sigma^q \to \{0,1\}$, and is said to be satisfied by an assignment $A : \Pi \to \Sigma$ iff $\varphi(A(\pi_{i_1}), \ldots, A(\pi_{i_q})) = 1$. In the optimization version of $q-\mathrm{CSP}(\Sigma)$, the goal is to find an assignment for the variables that satisfies as many constraints as possible. In the $(1, \varepsilon)$-gap version of this problem, the goal is to decide between

- There is an assignment $A : \Pi \to \Sigma$ satisfying all constraints.
- Every assignment $A : \Pi \to \Sigma$ satisfies at most an ε fraction of the constraints.

The PCP theorem is equivalent to the following theorem.

Theorem 4 (Hardness of Approximating CSPs) *There exist constants $\varepsilon < 1$ and $q > 1$ such that the $(1, \varepsilon)$-gap version of $q-\mathrm{CSP}(\{0,1\})$ is NP-hard.*

Let us briefly outline the translation between Theorem 4 and Theorem 1. The proof is equivalent to an assignment to the variables Π of a CSP instance.

Theorem 4 $\Rightarrow$ Theorem 1: The verifier computes the reduction from NP to the CSP problem given by Theorem 4. Then, the verifier uses its randomness to choose a uniform constraint, queries the values of its variables from the proof and accept according to the predicate.

Theorem 1 $\Rightarrow$ Theorem 4: Convert the verifier given by Theorem 1 into a CSP instance by enumerating over the random strings of the verifier. This gives a polynomial-sized set of q-ary constraints over the variables.

There are three parameters participating in this theorem:

- q - the arity of the constraints, this corresponds to the number of queries.
- ε - the maximal fraction of satisfied constraints in the "no" case, this corresponds to the soundness error.
- Σ - the alphabet. The number of bits read from the proof is thus $q \log |\Sigma|$.

Of course the parameters are allowed to be functions of the input size. Theorem 2, in this language, gives

Theorem 5 *For any $t(n) \leq O(\log n)$ the $(1, \exp(-\Omega(t(n))))$-gap version of $t(n)-\mathrm{CSP}(\{0,1\})$ is NP-hard.*

In these terms the parallelization question becomes

Is there a reduction from $t-\mathrm{CSP}(\{0,1\})$ to $O(1)-\mathrm{CSP}(\{0,1\}^t)$ maintaining the same gap ?

If there were such a generic reduction that works for any value of $t(n) \leq \log n$ and with $\varepsilon \approx 1/t(n)^{\Omega(1)}$ (where n denotes the input size), then together with Theorem 2 we would get that every NP-language can be verified using $O(\log n)$ random bits, reading $O(1)$ proof symbols over an alphabet of size $|\Sigma| \leq t(n)$ with soundness error at most $1/t(n)^c$. This is known as the "sliding scale conjecture" of [BGLR93], that appeared also earlier in [LY94].

Conjecture 1 ([BGLR93, LY94]) *There exists a constant $q \geq 2$ such that for every $\varepsilon(n) \geq 1/n^{O(1)}$, there is an alphabet $\Sigma(n)$, $|\Sigma(n)| \leq 1/\varepsilon(n)^{O(1)}$, such that the $(1, \varepsilon(n))$-gap version of $q-\mathrm{CSP}(\Sigma(n))$ is NP-hard. Equivalently,*

$$NP \subseteq PCP_{1,\varepsilon(n)}[O(\log n), q]_{\Sigma(n)}\,.$$

Here is the current status regarding this conjecture. The conjecture is known to hold for all values of $\varepsilon(n) \geq 2^{-(\log n)^{1-\delta}}$ for constant $\delta > 0$. For some fixed δ (possibly $\delta = 2/3$) it seems that one can even get this result with only three queries[2] ($q = 3$). For arbitrarily small constant $\delta > 0$ the number of required queries is $q = (1/\delta)^{O(1)}$, see [AS97, RS97, DFK+99]. Similar results with *two* queries are known only under quasi-NP-hardness assumptions [Raz98], and this question in general appears to be more difficult, as discussed below.

For smaller values of ε the conjecture remains open. The most interesting parameters are polynomially small soundness error, i.e., $\varepsilon = 1/n^{\Theta(1)}$.

There are two main approaches for achieving parallelization. The first is parallel repetition, which is a simple and straightforward combinatorial approach. Analyzing this simple construction is intricate and is the celebrated parallel repetition theorem of [Raz98]. The second approach is algebraic, and involves taking low degree curves in a low degree extension of the original system.

1.4 Two Queries

Two is the absolute minimal number of queries for which Conjecture 1 is possibly true. This question draws considerable interest both because of its optimality but even more so because of the wide applicability of 2-query PCP-systems (alternatively, 2-CSPs) to hardness of approximation. There is an impressively long list of hardness of approximation results that follow a paradigm initiated in [BGS98] and [Hås01]. These constructions compose an outer verifier and an inner verifier. The outer verifier is a 2-query PCP system (with an additional property called projection). The soundness of this verifier must be arbitrarily close to zero, and the alphabet size affects the overall size of the construction. The asymptotic tradeoff between the alphabet size and soundness error sometimes affects the approximation factor for which hardness is proven (although often these only appear in the second order term).

An important and powerful technique in the construction of PCPs is composition. Direct application of composition usually costs at least one additional query, and is thus unacceptable when the goal is a two-query PCP.

An exciting recent work of [MR08] gives a two-query PCP with sub-constant soundness error. The key technical contribution of this work is a novel method for composition that does not cost an additional consistency query. Thus, the number of queries remains two throughout. This result implies NP-hardness for a variety of approximation problems for which only quasi NP-hardness was known before (i.e. hardness under the stronger assumption of $NP \subseteq DTIME(n^{\text{poly} \log n})$). The PCP of [MR08] has a sub-optimal tradeoff between ε and Σ. Namely, the alphabet size is exponential in $1/\varepsilon^{O(1)}$ rather than the desired polynomial in $1/\varepsilon^{O(1)}$.

1.5 Organization

In the remainder of this manuscript we describe the two basic constructions of PCP systems with low soundness. In Section 2 we describe the construction based on parallel repetition, and in Section 3 the construction based on low degree extension. In Section 4 we discuss composition and the subtleties of applying it while preserving low soundness error.

[2]This can be attributed to folklore. We were unable to find a verified reference, but a related result is [Tar96].

2 Parallel Repetition

The simplest way to achieve parallelization is by simulating sequential repetition in parallel. Recall the straightforward amplification described in Theorem 2: We begin with a basic verifier that makes (say) two queries, always accepts yes instances, and rejects no instances with probability at least $1 - \varepsilon > 0$. The new verifier simulates t runs of the basic verifier, and makes all of the necessary queries to the proof, which amount to $2t$ queries.

What would be a parallelized version of this simulation? We move to a large alphabet in which one symbol encodes t answers. The new "parallelized" verifier will simulate t runs of the basic verifier, but instead of accessing the proof in $2t$ locations this verifier will only query two locations, one supposedly containing the t answers to the t first queries and the other containing the t answers to the t second queries. In other words, we "bunch" together t-tuples of symbols from the original proof, and we do so essentially for every possible t-tuple. (So the length of the new proof is the t-th power of the old length).

In case there was a proof causing the original verifier to accept with probability 1, then answering honestly according to this proof will cause the new verifier to accept with probability 1. However, it is surprisingly difficult to analyze the case where the original verifier accepted all proofs with probability at most ε. This is done in Raz's parallel repetition theorem [Raz98].

Raz's theorem is not stated in terms of parallelization of a PCP system but rather in terms of a two prover game that is played t times in parallel. In a two prover game a verifier randomly selects two questions, and sends one to each prover. The two provers can coordinate strategies beforehand, but cannot communicate once they get their questions. They each send an answer back and then the verifier accepts or rejects. Of course the goal of the provers is to cause the verifier to accept.

Let us explicitly make the translation into our setting. The two prover game corresponds to a special[3] kind of two-query PCP, in which the proof is divided into two parts, and the verifier's first query is always into the first part, and the second query is into the second part. We imagine each prover controlling one proof part, and answering according to it.

Playing t rounds of a two-prover game in parallel means that the verifier sends t questions to each prover (chosen by running the original game t times independently), and then each prover returns t answers. The verifier accepts if and only if it would have accepted every pair of answers in the original game.

What does the analogous t-parallel PCP system look like? Again the proof will have two parts and the verifier will make two queries, one into each part. The first query corresponds to a t-tuple of first queries, so the first part of the new proof will have an entry for every possible t-tuple of entries in the old first part. Similarly for the second part.

What is the soundness error of this new verifier? At first sight the answer might appear to be ε^t. However, this is false, and analyzing the true behavior turns out to be highly nontrivial. Let us try to understand why. For every possible choice of t queries, the new proof will have an entry *supposedly* consisting of t answers in some fixed proof for the original verifier. If the input is a "yes" instance then answering according to one fixed accepting proof is a good idea, and the verifier will always accept. Moreover, for a no instance, such a proof will be accepted with probability at most ε^t. The problem is that there may be other proofs that are accepted with higher probability. In such proofs the t answers will not always be consistent with one fixed original proof. Rather, one answer to a query may depend on the other $t - 1$ queries that are asked together with it. The "adversarial prover" receives more information in t queries than in one query, and may use this

[3]It is easy to convert the verifier given by Theorem 1 into having this form.

information advantageously.

As mentioned above, Raz's parallel repetition theorem [Raz98] implies that the effect of the extra information given by revealing all t questions at once rather than just one can be controlled, and the soundness error is upper bounded by $\varepsilon^{\Omega(t)}$.

Combining the PCP theorem together with the parallel repetition theorem, we get,

Theorem 6 *For any* $t : \mathbf{N} \rightarrow \mathbf{N}$,

$$NP \subseteq PCP_{1,\exp(-t(n))}[\,O(t(n) \cdot \log n)\,,\; 2\,]_{\{0,1\}^{O(t(n))}}\,.$$

What are the implications of this theorem in relation to the sliding scale conjecture (Conjecture 1)?

- For any constant t, this gives a verifier that makes 2 queries, reads $O(t)$ proof bits, and has soundness error at most $\exp(-t)$. In other words, the alphabet has size $2^{O(t)}$ and the soundness error is at most $2^{-\Omega(t)} = (1/|\Sigma|)^c$ for a fixed constant $c > 0$. So the sliding scale conjecture holds (with 2 queries!) for all constant values of t.

- For non-constant values of $t : \mathbf{N} \rightarrow \mathbf{N}$ (i.e. where t grows as a function of the input size) the number of random bits used by the parallelized verifier is $\Omega(t(n)\log n) \neq O(\log n)$. Unlike the sequential repetition case (Theorem 2) where the randomness can be reduced back to $O(\log n + t(n))$, here such derandomization cannot follow from simple hitting techniques. One reason is that while in Theorem 2 the proof size remained the same, here the proof length increases exponentially in t. We discuss derandomization further below.

- What if we allow slightly super-polynomial sized constructions? We get a verifier for NP that uses $O(t(n)\log n)$ random bits. This means that the proof length is $n^{t(n)}$ and indeed no longer polynomial. In order to understand the implications of this version of the theorem for hardness of approximation, it is more convenient to move to the language of CSPs. In this language, Theorem 6 is equivalent to the existence of a reduction from any NP-language to the $(1, \exp(-t(n)))$-gap version of $2{-}\mathrm{CSP}(\{0,1\}^{O(t(n))})$. Note however that this reduction runs in time $n^{O(t(n))}$.

  In order to figure out what kind of approximation algorithms are ruled out (under various hardness assumptions) we must express the functions $\varepsilon = \exp(-t(n))$ and $|\Sigma| = 2^{O(t(n))}$ in terms of the size N of the resulting CSP, rather than the input size n. Recall that the relation between the two is simply $N = n^{O(t(n))}$.

  Let us begin with $t(n) = \log n$ for example. We have $\varepsilon(n) = \exp(-t(n)) = 1/n$. However, as a function of N, this becomes $\varepsilon(n) = 2^{-\sqrt{\log N}}$ (since $\log N = \log(n^t) = t \log n = (\log n)^2$).

  Another interesting setting of the parameters is when $t = (\log n)^K$ for an arbitrarily large K. Assuming $NP \not\subseteq DTIME(n^{\mathrm{poly}\log(n)})$ Theorem 6 rules out the existence of a polynomial-time algorithm that decides the $(1, \exp(-\log n^{1-\delta}))$-gap version of $2{-}\mathrm{CSP}(\{0,1\}^{\log n^{1-\delta}})$ for $\delta = 1/(K+1)$.

  More generally, writing $t(n) = \log N / \log n$ we see that the soundness error is

  $$\exp(-c \cdot t) = \exp(-c \log N / \log n) \gg N^{-c},$$

  and will always be asymptotically more than polynomially small in N. We remain a step away from proving the sliding scale conjecture, even under increasingly strong hardness assumptions.

Derandomization. One possible way to improve the parallel repetition construction and hope to prove Conjecture 1 is by "derandomization". We mentioned before that the length of the new proof in the parallel repetition construction is the t-th power of the original proof length. But what if the verifier only ever looks at some (cleverly chosen) subset of the proof? A priory it may still be possible to prove that the soundness error remains small. For instance, in the gap amplification proof of the PCP theorem [Din07], the amplification step may be viewed as such a derandomization of the parallel repetition construction. However, this step only succeeds when the soundness error is quite large to begin with. In the current setting, when we start with a constant soundness error and wish to reduce it further, the technique no longer works, as demonstrated by [Bog05]. More generally, Feige and Kilian proved [FK95] that derandomization of parallel repetition cannot obtain the optimal parameters. They showed an algorithm that fools any derandomized verifier into accepting with probability that is asymptotically much larger than $\varepsilon^{\Theta(t)}$. (This probability depends on other parameters of the original verifier and can be as large as a constant for some verifiers!).

3 The Algebraic Approach to Parallelization

For any $t(n) \leq \log n$ Theorem 2 implies that every NP-language can be verified using $O(\log n)$ random bits, reading at most $t(n)$ proof bits, with soundness error $\exp(-\Omega(t(n)))$. Parallelizing this verifier would mean converting it to a verifier that reads only $O(1)$ proof symbols over an alphabet with size $2^{O(t(n))}$, and with the same soundness error.

Let us sketch the algebraic approach for parallelization. As explained earlier, the entire problem of parallelization boils down to consistency. The idea is to extend the proof by adding algebraic data that spreads the information around and allows "strong" consistency checks. The steps in this construction are as follows.

1. **Low Degree Test.** The starting point in this construction is the verifier from Theorem 2, which we denote by V_2. The proof for V_2 is embedded into a set $H^d \subset F^d$ and a low degree interpolation polynomial $p : F^d \to F$ is computed. The verifier V tests that p has low degree (using $O(1)$ queries).

2. **Parallelization through Curves.** The verifier V simulates V_2 but instead of evaluating p on t points, V computes a low degree algebraic curve that passes through these points and in *one* query reads a univariate polynomial supposedly equal to the value of p restricted to this curve. V checks that this polynomial is consistent with p on a random point. If not it rejects, and otherwise it simulates V_2 on the values from the curve and accepts/rejects according to what V_2 would have done.

Below we give some more details about these steps. However, we already mention that these steps alone will not get us to our goal. The reason is that the resulting verifier will read more than $O(t(n))$ proof bits for achieving error $\exp(-t(n))$. Nevertheless, the alphabet size can be reduced by combining the above together with a "composition" step. This will be the subject of Section 4.

3.1 Step 1 - Low Degree Extension and Low Degree Test

The low degree extension of a string $s \in \{0,1\}^n$ is a Reed-Muller encoding of it: Let F be a finite field, fix a subset $H \subset F$, and assume $n = |H|^d$. We identify the indices $\{1, \ldots, n\}$ with the elements in H^d. Now the string s can be viewed as a function $s : H^d \to \{0,1\}$ or even as $s : H^d \to F$.

The low degree extension of s (with degree $h = |H| - 1$) is defined to be the unique polynomial $e : F^d \to F^d$ of degree h in each of its d variables such that

$$\forall x_1, \ldots, x_d \in H, \quad e(x_1, \ldots, x_d) = s(x_1, \ldots, s_d).$$

Its existence and uniqueness follow from basic facts about interpolation.

The first step is syntactic. V now expects the *low degree extension* of a proof for V_2. Now the proof bits are no longer enumerated 1 through n but rather indexed by the elements of H^d. We also add more proof entries for all the elements of F^d, and the proof is now identified with a function $p : F^d \to F$ (and already the alphabet increased from binary to $\Sigma = F$). The verifier V_2 reads some t bits from the proof, so this corresponds to evaluating p at t points in F^d.

The nice thing about a low degree function is that in some sense its evaluation at every point in F^d gives a little information about its evaluation at any other point. For example, the restriction of p to a line must have low degree, and so cannot obtain any arbitrary $|F|$-tuple of values. This gives us a lot of power in testing consistency.

The first step is called a low degree test: the verifier checks that this proof p is a *low degree* function, using few queries. How is this done? there is a long list of works on low degree testing. In the case of small soundness error and constant number of queries there are two basic analyses: the line-vs.-point test of [AS97] and the plane-vs.-point test of [RS97]. For example, in the line-vs.-point version the proof also consists of an entry for each line (a line in F^d is defined by two points $\mathbf{x}, \mathbf{y} \in F^d$ and is the function $\ell : F \to F^d$ defined by $\ell(\alpha) = (1 - \alpha)\mathbf{x} + \alpha\mathbf{y}$), and in that entry the proof specifies a univariate polynomial of degree at most dh that is supposed to equal $p \circ \ell$, the restriction of p to the line. This is sometimes called a "lines table". The test is very simple:

- Select a random line $\ell : F \to F^d$, and a random point $\alpha \in F$. Compute $\mathbf{x} = \ell(\alpha)$.
- Read (from the lines table) the univariate polynomial p_ℓ that supposedly equals $p \circ \ell$.
- Read $p(\mathbf{x})$.
- Accept iff $p(\mathbf{x}) = p_\ell(\alpha)$.

If p is a low degree function then a 'truthful' lines table would cause the verifier to accept with probability 1.

How would this test behave if p is not a low degree function? Ideally we would like the test to accept with probability no more than ε. In reality however, this depends on the distance of p from a low degree function. For example, suppose p is a low degree function that is randomly corrupted on $1 - \varepsilon$ fraction of the points. In such a case the test would accept with probability at least ε. Another instructive example is when p is a hybrid of $L = 1/\varepsilon$ distinct low degree polynomials $p_1, \ldots, p_L$. More precisely, we define p as follows. For each point $\mathbf{x}$ we select independently at random $i \in \{1, \ldots, L\}$ and set $p(\mathbf{x}) = p_i(\mathbf{x})$, and similarly for each line we select independently at random $i \in \{1, \ldots, L\}$ and set $p_\ell = p_i \circ \ell$. This p passes the test with probability about ε.

These examples highlight the subtlety of analyzing the low degree test in the case of small soundness error. Furthermore, they show us what we can possibly expect to conclude from the fact that the test passes with probability $\geq \varepsilon$. Indeed, one can prove (see [AS97, RS97]) that the combination of the two above 'counter-examples' is essentially the worst that can happen. In other words, whenever the test accepts with probability above ε, there must be a (short) list of low degree functions that agree non-trivially with p.

Moreover, it turns out that this seemingly weak conclusion of having a list of low degree functions that agree non-trivially with p is sufficiently powerful to be incorporated into a low soundness error PCP system.

3.2 Step 2 - Restriction to Curves

Having checked that p has low degree, the task faced by our verifier is to simulate V_2 using only $O(1)$ queries. Now V_2 reads $t \gg 1$ points $\mathbf{x}_1, \ldots, \mathbf{x}_t \in F^d$ in the proof $p : F^d \to F$. The idea is to "bunch" these points together by passing a low degree curve through them. Select $\mathbf{x}_0 \in F^d$ uniformly at random and let $\gamma : F \to F^d$ be a random curve of degree t such that $\gamma(i) = \mathbf{x}_i$ for $i = 0, \ldots, t$ (we assume $\{0, 1, \ldots, t\} \subset F$). Now our proof will also consist of a "curves-table", i.e. of an entry for each such choice of curve γ. The value in the γ-th entry will supposedly specify a univariate polynomial of degree $t \cdot dh$ that is supposed to equal $p \circ \gamma$, the restriction of p to the curve γ. The verifier will perform the following test

- Simulate V_2 and compute a list of t points $\mathbf{x}_1, \ldots, \mathbf{x}_t$ in which to query p. (But do not make these queries as we cannot afford so many of them).
- Select a random point $\mathbf{x}_0 \in F^d$ and compute the curve γ passing through $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t$. Read the entry p_γ corresponding to γ and check that the values $p_\gamma(1), \ldots, p_\gamma(t)$ would have caused V_2 to accept. If not, reject.
- Choose a random $\alpha \in F \setminus \{1, \ldots, t\}$, compute $\mathbf{y} = \gamma(\alpha) \in F^d$ and read $p(\mathbf{y})$. Accept iff $p_\gamma(\alpha) = p(\mathbf{y})$.

As in the case of the low degree test, it is easy to see that an honest proof for V_2 can be extended to cause the new verifier to accept with probability 1.

Also, in case every proof is accepted by V_2 with probability below ε (i.e., a no instance), the new verifier will also accept with similar probability. The reason is as follows. Either $p_\gamma = p \circ \gamma$ in which case the simulation is faithful and V behaves like V_2. Otherwise, $p_\gamma \neq p \circ \gamma$ but then the consistency test on a random point on the curve γ will fail with high probability.

3.3 Modifying step 2 for two queries

At this point the described verifier makes 4 queries: two for the low degree test, and two for the curve consistency check. Let us outline how to combine these two tests together thereby reducing the total number of queries to two.

The low degree test step requires two queries: a point and a line (or alternatively: a point and a plane). The curve step also requires two queries: a curve and a point. It is possible to reuse the same point in both parts, therefore reducing the number of queries to 3.

We can further reduce the number of queries to two. One way is by considering a hybrid object that consists both of a line and of a curve. Basically, we consider functions $\gamma : F^2 \to F^d$ (these are two-dimensional varieties) that contain as subsets all points of a specified curve and all points of a specified line. A similar construction is described in [MR08, Section 3.3], called a Manifold vs. Point construction.

The proof will then consist of the point evaluation of p as before, and instead of having a lines-table and a curves-table there will be one curves table whose entries supposedly contain the restriction of p to these 2-dimensional varieties. The verifier will again choose γ that passes through the t points required by V_2 and check consistency with p on a random point of γ.

The alphabet size is only quadratically larger compared to the previous subsection. Unfortunately, both of these sizes are exponential in $1/\varepsilon^{O(1)}$ instead of the desired polynomial in $1/\varepsilon^{O(1)}$. This can be improved, as described in the next section.

4 Composition

The verifier described in the previous section has soundness error of $\varepsilon \leq 1/|F|^c$ for some constant $c > 0$. However, the alphabet size is on the order of $\exp(1/\varepsilon^{O(1)})$ rather than the desired $1/\varepsilon^{O(1)}$, since a symbol needs to specify the coefficients of a polynomial whose degree is some constant power of $|F|$.

Fortunately, the alphabet size can be reduced by composition and recursion [AS98]. The idea is to simulate the verifier, figure out which proof bits it needs to read, but then instead of performing the verification, we let another "inner" verifier verify that *this input would have caused the original verifier to accept*. This inner verifier will not read the entire input but only a small (random) part of it, relying on an additional proof for the task. Thus the verification task can potentially rely on reading even fewer proof bits than before. Since the input for the inner verifier is much smaller than the original input, it is allowed to be less efficient and so potentially easier to construct.

The usual way to apply composition costs (at least) one additional query. This query essentially makes sure that the different invocations of the inner verifier are consistent with each other. This means that the number of composition steps factors into the total number of queries. Each composition step also increases the size of the construction, and that poses another limitation to applying composition many times. The best results achieved in this framework are

Theorem 7 ([RS97, AS97, DFK$^+$99]) *For all constant $\delta > 0$,*

$$NP \subseteq PCP_{1,\exp(-(\log n)^{1-\delta})}[O(\log n), O(1)]_{\{0,1\}^{O((\log n)^{1-\delta})}}$$

This settles Conjecture 1 for all values of $t(n) \leq \log n^{1-\delta}$ for arbitrarily small constant $\delta > 0$. In the next subsection we describe some of the ingredients that go into the proof of this theorem.

4.1 A Verifier that also decodes

We now describe the composition step that goes on in the proof of Theorem 7. Composition is somewhat more subtle in the low soundness case. The starting point is a verifier as described in Section 3 which we call in this context an "outer verifier". This outer verifier makes two (random) queries that are highly asymmetric. The first query is answered by an element in F and its size is at most $1/\varepsilon^{O(1)}$ (where ε denotes the soundness error). The second query is answered by a low degree polynomial and its description size is exponential in $1/\varepsilon^{O(1)}$. The outer verifier checks that (i) the second query satisfies some predicate and that (ii) it is consistent with the first query. Consistency means that the evaluation of the polynomial at one of its points should equal the answer of the first query.

A Decoding Verifier. Rather than reading the second query, the outer verifier will run a *decoding verifier* for this job. The decoding verifier will expect an encoding of the answer to the second query over a smaller alphabet and will be able to

- Test that the encoding is valid (local testability)
- Decode a value from the encoded message (conditional local decodability)

The decodability is called *conditional* since the verifier is allowed to reject[4] if it finds an inconsistency.

[4]This ability to reject makes this object much easier to construct compared to a locally decodable code (LDC).

More precisely, the decoding verifier is given an input x and instead of a witness w it gets oracle access to a special encoding π of w. (In our example x is the predicate from the outer verifier that specifies which are the valid answers to the second query; w is an answer to the second query; and π an encoding of w over a smaller alphabet). The decoding verifier is also given an index (or indices) of symbols from w. The verifier reads x, probes π in a small number of locations, and is supposed to output the value of w on the specified indices (i.e., decode π), or reject.

Thus a decoding verifier is just like a regular verifier except that in addition to accepting/rejecting it also outputs an answer. This answer will enable the outer verifier to perform the consistency test.

The verifier is required to always output the correct values if π is a valid encoding, and otherwise to err with probability at most ε. The definition of an erroneous outcome is slightly tricky: we consider a list of all messages whose valid encodings are nontrivially correlated to π (i.e., a "list-decoding" of π). The output of the verifier is considered an *error* only if it agrees with none of the messages in this list. This seemingly tolerant list-decoding definition is necessary. Just as in the situation of the low degree test (see Section 3.1), one can easily generate 'counterexamples' such as a hybrid of $1/\varepsilon$ valid encodings that demonstrate the inherent need for list-decoding.

Our description of a decoding verifier is intentionally loose. A nice formal definition of such an object has been recently put forth in [MR08] under the name LDRC (locally decode / reject code). Previous works on small soundness error PCPs implicitly described such verifiers using definitions that were more ad-hoc.

Composition. In summary, the composed verifier will begin by simulating the original verifier and read the first query from the original proof (this is an element in F). Then, instead of reading the second query from the proof, it will run the decoding verifier on an encoding of the answer to the second query, specifying the required message bits that will enable the consistency check against the first query. The verifier will accept only if the decoding verifier did not reject and if its output is consistent with the answer to the first query. The total number of queries made by the composed verifier is equal to the number of queries made by the decoding verifier plus the additional first query of the outer verifier (this is the so-called consistency query).

How do we construct a decoding verifier? We list two main constructions.

- **Curves.** The verifier described in Section 3 can be modified into a decoding verifier. Recall that the underlying encoding is as follows: A message is encoded by its low-degree extension polynomial p evaluated at all points of F^d plus a curves table consisting of the restriction of p to a certain collection of low-degree curves (here we use the word curves to also include lines and two-dimensional varieties). The local testability has already been described in Section 3. How will the verifier decode? Suppose it is given as input a point $\mathbf{x} \in F^d$ and the output should be $p(\mathbf{x})$. The verifier selects at random a curve that passes through the point $\mathbf{x}$, and selects another random point $\mathbf{y}$ on the curve. It then reads the entry corresponding to this curve from the proof, evaluates it at $\mathbf{x}$ and at $\mathbf{y}$. It rejects if the value at $\mathbf{y}$ is inconsistent with $p(\mathbf{y})$ and otherwise outputs the value at $\mathbf{x}$.

- **Hadamard.** We will not describe the details of this construction, just mention that it is based on a variant of the Hadamard Code over a non-binary field. The length of this encoding is exponential in the message length, but the advantage of this construction is that the alphabet size is simply the field F.

The proof of Theorem 7 involves several steps of composition. First a number of steps involving the "curves" decoding verifier are applied, and then when the alphabet size is small enough the final composition step is with the "Hadamard" verifier.

4.2 A New 2-Query Composition with Small Soundness

An new method for composition has been recently developed in [MR08]. They show how to compose two decoding verifiers in a way that maintains low soundness error and avoids the need for an extra consistency query. Thus, they are able to get a two-query PCP with sub-constant soundness, and polynomial (in fact, nearly linear) proof length.

How do they avoid the need for an extra consistency query? The idea is to parallelize the consistency query with one of the queries of the inner verifier. Of course doing this in a straightforward way is doomed to fail, and one needs to add more "confusing" information to maintain soundness. The details are beyond our scope to describe.

4.3 Why composition of PCPPs fails in the small soundness case

Clearly a "decoding verifier" is a stronger construct than a regular PCP verifier since it is required to also decode. By stronger it is meant that the first construct implies the second, but not necessarily vice versa.

Is decoding really necessary for composition?

A related construct that turns out to lie between a verifier and a decoding verifier is the verifier of proximity aka assignment tester. PCPs of Proximity (PCPPs) [BGH$^+$06] or assignment testers [DR06] give a nice modular way to think of composition. One nice feature in their composition is that it does not require an additional consistency query. In fact, the number of queries does not increase at all. However, as we will see below, such composition seems to be inherently limited to the case of constant soundness error (that does not tend to zero).

Just as a decoding verifier, the verifier of proximity V is interested in x, w, π. x is the input, w is a witness for membership of x in L, and π is an additional proof. Now the verifier V doesn't need to decode symbols from w using π. Rather, the verifier's job is to test that w is a valid witness for $x \in L$ (using π). Of course the verifier cannot read the entire w and instead it gets oracle access to w. The number of queries made by V is the total number of symbols it reads from w and π. This is why V can really only check *proximity* of w to a valid witness.

It should be clear that a decoding verifier is easily made into a verifier of proximity: Proximity is checked simply by decoding a random index in w using π and then comparing it to the corresponding symbol in w.

How do we use a verifier of proximity as an inner verifier in composition? First simulate the outer verifier on a random string r. This simulation generates some indices $i_1, \ldots, i_q$ in which the verifier needs to access the proof. Now, we invoke the verifier of proximity with the witness w being the sub-proof specified by the indices $i_1, \ldots, i_q$. The inner verifier also has oracle access to an additional proof π. It is supposed to accept if w would have caused the original verifier to accept, and reject otherwise.

Of course the last part is inaccurate as we may only expect the inner verifier to reject with good probability if w was noticeably far from any valid witness. For this to work we need the outer verifier to have an additional property called *robust soundness*. Regular soundness guarantees that on a no instance the verifier will reject with some probability $1 - \varepsilon$. Robust soundness means that with this probability the verifier will "robustly" reject. By robustly rejecting it is meant that the answers to the verifier's queries are far from accepting: one needs to change at least a ρ fraction of these answers in order to make the verifier accept. In other words, with probability $1-\varepsilon$ the local view of the verifier is ρ-far from an accepting view. For the composition to work we need the robustness of the outer verifier to match the proximity of the inner verifier, i.e. $\rho \geq \delta$.

The reason that this scheme breaks down in the small soundness error case is because of the limitations on the robustness parameter ρ. In all known constructions, always $\rho \leq 1/q \leq 1/2$ where q is the number of queries made by the outer verifier. This limits $\delta \leq \frac{1}{2}$. On the other hand, it is always the case that $\varepsilon \geq (1-\delta)^{q_{in}}$ where q_{in} is the number of queries made by the inner verifier. Indeed, think of a valid proof in which each bit is flipped independently with probability δ. With probability $(1-\delta)^{q_{in}}$ the inner verifier will only see non-flipped values and must accept, so $\varepsilon \geq (1-\delta)^{q_{in}}$. This means that ε cannot tend to zero as long as δ is bounded away from 1.

5 Conclusion

We have sketched PCPs with small soundness error that go towards proving the sliding scale conjecture. We also saw the related parallelization question. It remains open whether there is a PCP verifier that makes a constant number of queries (ultimately, two) over an alphabet of polynomial size, and has an inverse polynomial soundness error. In fact, we feel that trying to refute this conjecture does not seem unreasonable at this point.

A slightly easier question would be to come up with a low degree test with similar parameters. In other words, a verifier for the fact that a given function has low degree, such that it reads a constant number of queries over a polynomial-sized alphabet, and has inverse polynomial soundness error. More generally, we wonder if one can construct locally testable codes with such parameters.

Acknowledgement

We thank Dana Moshkovitz for helpful discussions.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[AS97] S. Arora and M. Sudan. Improved low degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4–6 May 1997.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[BGH+06] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SICOMP Special Issue on Randomness and Computation*, 36(4):889–974, 2006.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient multi-prover interactive proofs with applications to approximation problems. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 113–131, 1993.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915, June 1998.

[Bog05] A. Bogdanov. Gap amplification fails below 1/2. Comment on ECCC TR05-046, can be found at `http://eccc.uni-trier.de/eccc-reports/2005/TR05-046/commt01.pdf`, 2005.

[DFK$^+$99] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra. PCP characterizations of NP: Towards a polynomially-small error-probability. In *Proc. 31st ACM Symp. on Theory of Computing*, 1999.

[Din07] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3), 2007.

[DR06] I. Dinur and O. Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SICOMP Special Issue on Randomness and Computation*, 36(4):975–1024, 2006.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *Journal of the ACM*, 43(2):268–292, 1996.

[FK95] U. Feige and J. Kilian. Impossibility results for recycling random bits in two-prover proof systems. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 457–468, 1995.

[Hås01] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[MR08] D. Moshkovitz and R. Raz. Two query PCP with sub-constant error. In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, 2008. To appear.

[Raz98] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, June 1998.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 475–484, 1997.

[Tar96] G. Tardos. Multi-prover encoding schemes and three-prover proof systems. *Journal of Computer and System Sciences*, 53(2):251–260, 1996.