Talk given at workshop on Algebraic Methods in Complexity Theory

On the role of algebra in the efficient verification of proofs

Madhu Sudan *

Abstract

This article extracts the elements of algebra that play a central role in the design of efficient probabilistic verifiers or "probabilistically checkable proof systems (PCPs)". The main algebraic elements are low-degree polynomials over finite fields. Their role can be broken up into three essential elements:

1. Their classical role in the design of error-correcting codes.
2. Their recently discovered property of being efficiently locally checkable.
3. The existence of characterizations via polynomials of fundamental complexity classes including NP, PSPACE and NEXPTIME.

1 Introduction

The reader of this writeup is assumed to be slightly familiar with the series of developments in complexity theory relating to proof checking over the last five years. These developments started with the interactive proofs for $\sharp\mathrm{P}$, and PSPACE due to Lund, Fortnow, Karloff and Nisan [28] and Shamir [35] respectively. The developments have gone on to provide different proof checking characterizations of other complexity classes including NEXPTIME, due to Babai, Fortnow and Lund [6] and NP, due to Arora and Safra [3] and Arora, Lund, Motwani, Sudan and Szegedy [2]. The effect of these characterizations was highly amplified by the connection to world of combinatorial optimization discovered by Feige, Goldwasser, Lovasz, Safra and Szegedy [17]. They turned these characterizations of NEXPTIME and NP into hardness result for the approximability of the clique size in graphs. Further notable successes along these lines came in the works of Bellare and Rogaway [10] and Feige and Lovasz [19] and Arora, Lund, Motwani, Sudan and Szegedy [2]. For further details, the reader is referred to the survey articles by Babai [4], Goldreich [24] or Johnson [26].

The motivation of this writeup is to focus on certain algebraic elements used in the construction of these efficiently checkable proofs (or more correctly - the efficient verfication of proofs) and justify their participation in the construction. The writeup provides neither a complete proof nor a road-map of the proofs of these characterizations. Instead, it provides an alternate perspective on the developments by laying out an analogy with some classical techniques in coding theory. This perspective gives a glimpse into the intuition underlying the complicated proofs without getting too deeply into the technical details that are involved. Hopefully this intuition will help the more motivated reader in understanding the complete proofs of the various results described here, which may be found in the original papers – or alternatively in the dissertations of Arora [1] or this author [37].

*IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, U.S.A. email: madhu@watson.ibm.com.

2 Probabilistically checkable proofs

We start by describing the traditional connection between proof checking and complexity theory. This connection is fundamental to the definition of the complexity class NP.

Definition 1 (NP) *A language L belongs to NP if there exists a polynomial time non-deterministic Turing machine V such that for all $x \in \Sigma^*$*

- *$x \in L \Rightarrow$ there exists a sequence of non-deterministic choices (or witness or proof) π such that V on choice π accepts the input x.*
- *$x \notin L$ implies for no choice π will V accept x.*

In terms of the above definition, the statements $x \in L$ are our potential theorems. If the assertion is true, then there must exist a short proof π. If the assertion is not true then no proof π works. The next definition is a culmination of a sequence of efforts which owe their origins to the work of Goldwasser, Micali and Rackoff [25]; Babai and Moran [8]; Ben-Or, Goldwasser, Kilian and Wigderson [12]; Fortnow, Rompel and Sipser [20]; Babai, Fortnow, Levin and Szegedy [7]. The notion of a probabilistic oracle machine used below was introduced by Fortnow, Rompel and Sipser [20], and the parameterization of the resources used by the verifier was implicit in the work of Feige, Goldwasser, Lovasz, Safra and Szegedy [17] and explicit in the following defintion of Arora and Safra [3].

Definition 2 (PCP [3]) *For functions $r, q : \mathcal{Z}^+ \to \mathcal{Z}^+$, we define the class $\mathrm{PCP}(r, q)$ as follows: A language $L \in \mathrm{PCP}(r, q)$ if there exists a probabilistic polynomial time oracle machine V, accessing a polynomial sized oracle π, such that for all $x \in \{0, 1\}^n$:*

- *$x \in L \Rightarrow \exists \pi$ such that on all choices of random strings, the verifier V^π accepts x.*
- *$x \notin L \Rightarrow \forall \pi$, V^π rejects x with probability (over internal coin flips) at least 1/2.*

Moreover, V uses at most $r(n)$ coin flips for its probabilistic verification and queries the oracle π at most $q(n)$ times.

An obvious connection between NP and PCP is $\mathrm{NP} = \mathrm{PCP}(0, \mathrm{poly})$. This connection describes why PCP may be considered a natural variant of NP. However some very surprising and strong characterizations of NP are now available in terms of PCP.

The characterization below was proved by Arora, Lund, Motwani, Sudan and Szegedy [2]. They built upon an earlier characterization, due to Arora and Safra [3], which showed that $\mathrm{NP} = \cup_{k>0}\mathrm{PCP}(k \log n, \sqrt{\log n})$. All characterizations of this form owe their origin to the characterization of NEXPTIME due to Babai, Fortnow and Lund [6], which led to containment results of the form $\mathrm{NP} \subset \mathrm{PCP}(\mathrm{polylog}\, n, \mathrm{polylog}\, n)$ (due to Babai, Fortnow, Levin and Szegedy [7]) and $\mathrm{NP} \subset \mathrm{PCP}(\log n \,\mathrm{loglog}\, n, \log n \,\mathrm{loglog}\, n)$ (due to Feige, Goldwasser, Lovasz, Safra and Szegedy [17]). The work of Arora and Safra [3] gave the first exact characterization of NP in terms of PCP with low-query complexity.

Theorem 3 ([3, 2]) $\exists C < \infty$, *such that* $\mathrm{NP} = \cup_{k>0}\mathrm{PCP}(k \log n, C)$.

Let us try to understand the theorem above in terms of the impact on proof checking. It says that there is a way to "format" proofs, so that a reader of the proof need not read it everywhere in order to verify its correctness. In fact probabilistically sampling it at just C bits suffices to gain reasonable confidence in the correctness of the proof. The low randomness complexity indicates that the format on the proofs does not make it too large compared to the old proof. The new proofs are only polynomially larger than the old ones.

What makes these proofs work ......? Read on .....

3 Error-correcting codes

We motivate the need for coding theory by looking at a very special case of verifying Hamiltonicity.

Local Hamiltonicity: *Graph G and an edge e not in G, with a promise that $G+e$ is Hamiltonian. Decide if G is Hamiltonian.*

The knowledge that adding one edge e to G makes it Hamiltonian does not make it any easier to decide Hamiltonicity and this can be shown via the following assertion.

Proposition 4 **Local Hamiltonicity** *is* NP*-hard under Cook reductions.*

Let us now contrast the effect of Proposition 4 with the effect of Theorem 3. Suppose we have a PCP verifier V for Hamiltonicity (regular Hamiltonicity - not local), and suppose π is a proof which convinces V that $G+e$ is Hamiltonian. Suppose we consider the action of V^{π} on input G. In this case V rejects with probability at least $1/2$ if G is not Hamiltonian. Now if, in the case when G is Hamiltonian, V accepts the proof π with probability more than $1/2$, then V is effectively deciding Local Hamiltonicity (i.e., given evidence that $G+e$ is hamiltonian, V is able to decide if G is Hamiltonian), Since V is a polynomial time machine and deciding Local Hamiltonicity is NP-hard, it must be the case that V rejects π on input G with probability $1/2$ (independent of the Hamiltonicity of G).

But in the entire course of its actions V has only seen C bits of π. How is it possible that it is able to tell if π is good for G or not based on such limited sampling? We assert that it must be the case that π encodes the input G, so that for any two inputs G and G', the space of valid proofs π for G and π' for G' are far apart - so far that sampling π and π' at only C bits must reveal one place where $\pi(i) \neq \pi'(i)$. A reader who is familiar with coding theory will immediately see the connections with error-correcting codes - π is an encoding of G in an error-correcting code. We now define the notion of an error-correcting code formally and outline the parameters we will focus on.

We define codes over an alphabet Σ. For two strings σ_1 and σ_2 in Σ^n, the *(relative) distance* between the two, denoted $\Delta(\sigma_1, \sigma_2)$, is the fraction of indices i such that $\sigma_1^{(i)} \neq \sigma_2^{(i)}$. An $[n, k, \delta]$ code C over Σ is a mapping from Σ^k or *messages* to Σ^n or codewords, with the property that for any two message w_1 and w_2, the distance between the codewords $C(m_1)$ and $C(m_2)$ is at least δ. Thus $k \log |\Sigma|$ is the information content of a message. We refer to k as the message size and n as the block length of a code.

For our purposes it suffices to put in some broad restrictions on these parameters.

Definition 5 *A family of codes C_i with parameters $[n_i, k_i, \delta]$ over the alphabet Σ_i, is* good *if the following hold:*

- $k_i \to \infty$ *as* $i \to \infty$.

- *There exists a constant c such that* $n_i \leq k_i^c$.
- $\delta > 0$.

Let V be the PCP verifier obtained from Theorem 3 for the Hamiltonicity language. For every Hamiltonian graph G on n nodes, let $\pi_n(G)$ be a proof that convinces V with probability 1. Then the informal argument presented so far can be summed up as saying: "The mappings π_n must form a *good* code over the alphabet $\{0, 1\}$."

The oracles/proofs in the PCP systems offer an even more interesting feature in error-correcting codes, which seems to be a new addition to the study of error-correcting codes. Unlike the earlier case, we will not give any arguments as to why it is necessary to add this new ingredient to the codes we construct. Instead we will simply define the property and allow it to motivate itself.

The new notion is that of "local checkability" of a code. Informally a locally checkable code is one in which error-detection (i.e., determining if a given word W is a codeword) can be performed very efficiently probabilistically. The locality is parameterized by two parameters p (probes) and γ (error). The former parameter p is the number of letters of the received word that are scanned by the probabilistic error-detector. The latter parameter is the probability with which the error-detector detects errors if the received word is not uniquely decodable. The following formalism is due to Rubinfeld and Sudan [34].

Definition 6 ([34]) *For a positive integer p and a positive real number γ, an $[n, k, \delta]$-code C over the alphabet Σ is (p, γ)*-**locally checkable** *if the following exist*

- *A probability space Ω which can be* efficiently *sampled.*
- *Functions* $q_1, q_2, \ldots, q_p : \Omega \to \{1, \ldots, n\}$.
- *A boolean function* $V : \Omega \times \Sigma^p \to \{0, 1\}$.

with the property that for all $w \in \Sigma^n$, if

$$\Pr_{r \in \Omega}\left[V(r, w_{q_1(r)}, \ldots, w_{q_p(r)}) = 0\right] < \gamma$$

then there exists a (unique) string $m \in \Sigma^k$ such that $\Delta(w, C(m)) < \delta/2$. Conversely, if $w = C(m)$ for some m, then $V(r, w_{q_1(r)}, \ldots, w_{q_p(r)}) = 1$ for all $r \in \Omega$.

In the next section we will show how low-degree polynomials over finite fields, provide good codes. Even more interestingly we mention how they provide good codes that are $(2, 1/8)$ locally checkable over a polylogarithmic sized alphabet!

4 Polynomials over finite fields

In turns out that some of the simplest error-correcting codes are based on an easily proved property of polynomials over finite domains. This fact is that a polynomial of degree d is either identically zero or non-zero for most of its inputs provided the size of the domain we are working over is large compared to d. Before we state this fact formally we see how to use this fact to design constant distance codes over an alphabet which grows logarithmically with k and n which grows as a linearly with k.

We let $n_i = 2^i$ and $k_i = 2^{i-1}$. We choose as our alphabet a finite field F of size n and we let $\{\zeta_i | i \in \{1, \ldots, n\}\}$ denote an enumeration of the elements of F. Let the message represent a polynomial of degree $k-1$ over the field F. More precisely, let $m = m^{(0)} \ldots m^{(k-1)}$ represent the polynomial $M(x) = \sum_{i=0}^{k-1} m_i x^i$, where the m_i's are elements of F. The encoding $C(m) = C(m)^{(0)} \cdots C(m)^{(n-1)}$ is given by $C(m)^{(i)} = M(\zeta_i)$.

The code used above is a special case of a family of codes well-known in the coding theory literature as the Reed-Solomon codes (cf. [30]). The relative distance achieved by the code is $k/n = 1/2$ and this is proven by the following property of polynomials.

Theorem 7 (cf. [16, 36, 38]) *Let m be a positive integer and let $f, g : F^m \rightarrow F$, be distinct m-variate polynomials of total degree*[1] *at most d. Then f and g disagree in at least $d/|F|$ fraction of their inputs.*

Comment 1: Note that the number of variables, m, does not appear in the bound on the distance.

Comment 2: Notice that this is the distance measured over the alphabet F. It must be observed here that *large* distances are easy to achieve over *large* alphabets. Consider for instance two random binary strings – these are likely to differ in only half the places. But two random strings over the alphabet F differ in all but $\frac{1}{|F|}$ fraction of the places.

Comment 3: It is possible to translate the above into a binary code with reasonable distance. For instance, if we just represent every element of F by a $\log|F|$-bit binary string, then we can write the codewords of the Reed-Solomon codes as $|F|\log|F|$-bit strings, which have a distance of $\frac{1}{\log|F|}(1 - \frac{d}{|F|})$, which though not a constant, is nevertheless non-trivial.

The generality of Theorem 7, in that it works multivariate polynomials as efficiently as for univariate polynomials, indicates that we could as easily use multivariate polynomials to do our encoding.

For positive integers m,d, let $N(m, d)$ denote the number of monomials in m-variables of total degree at most d. [2] For $d \leq |F|$, we get the code $E_{m,d,F}$ as follows:

$$E_{m,d,F} : F^{N(m,d)} \mapsto F^{|F|^m}$$

where the message is viewed as specifying all the coefficients of a degree d, m-variate polynomial over F, and the encoding corresponds to the value of the polynomial on all possible inputs from F^m. Again the relative distance of such a polynomial is $(1 - \frac{d}{|F|})$ (using Theorem 7). We start by looking at a very important special case of such polynomial codes.

Hadamard Codes A special case of the $E_{m,d,F}$ codes is when $d = 1$ and $F = \mathcal{Z}_2$. In this case, $N(m, d) = m$. A message is thus given by m coefficients $c_1, \ldots, c_m$. Its encoding is the value of the polynomial $C(x_1, \ldots, x_m) = \sum_{i=1}^{m} c_i x_i$ at all m-tuples $x = (x_1, \ldots, x_m) \in \mathcal{Z}_2^m$. Schwartz's Lemma is still applicable here and guarantees that any two codewords differ in half the places! The nice thing about this code is that it is over a binary alphabet. The bad thing is that the codeword is 2^m bits long, which is exponential in the message size (which was m).

However, the Hadamard codes are already useful in getting use a *good* code over a constant alphabet size. We achieve this as follows: For $k_i = 2^{i-1}$ we use the Reed-Solomon code described above to form an

[1] The total degree of a monomial is the sum of the degrees of the variables in the monomial. The total degree of a polynomial is the maximum of the total degrees of the monomials that appear with a non-zero coefficient in the polynomial.

[2] Notice that $N(m, d) = \binom{m+d}{d}$.

"outer" code over an $n = 2^i$-ary alphabet. We then use the Hadamard code to encode each letter of the 2^i-ary alphabet. Thus the encoding of $i2^{i-1}$ bit message is a 2^{2i}-bit word which has distance $1/4$. In other words, we get a family of $[k^2, k, 1/4]$ codes over a binary alphabet. Not bad considering the simplicity of the techniques involved!

But a lot more interesting than the elementary applications of the Hadamard codes are their local checkability properties. We start to investigate them next.

4.1 Locally checkable codes via polynomials

We start with identifying the codewords in a Hadamard code with the degree 1 function $f : \mathcal{Z}_2^m \to \mathcal{Z}_2$ which describes the value of the codeword at the various locations. We use the term *linear* to denote such functions: i.e., functions of degree 1 with the constant term being zero. The fundamental property of linear functions which provides the intuition for the local checkability of Hadamard codes is the following:

Linearity Property: *$f : \mathcal{Z}_2^m \to \mathcal{Z}_2$ is a linear function if and only if*

$$\forall x, y \in \mathcal{Z}_2^m \quad f(x) + f(y) = f(x + y).$$

Very surprisingly the above property can be turned into one of a very "robust" form - which does not involve the "universal" quantifier above. This breakthrough was achieved by Blum, Luby and Rubinfeld [15] in their effort to find simple verification mechanisms for programs in the spirit of Blum and Kannan [14]. The exact bound specified below on Δ is stronger than that in the work of Blum, Luby and Rubinfeld, [15], and is due to Bellare, Goldwasser, Lund and Russell [9]. (In what appears below we generalize our notation slightly to use $\Delta(f, g)$ to denote the probability that two functions disagree - which is the same as the distance between the words corresponding to f and g.)

Theorem 8 (Linearity Test: [15]) *If a function $f : \mathcal{Z}_2^m \to \mathcal{Z}_2$ is a function which satisfies*

$$\Pr_{x,y \in \mathcal{Z}_2^m} [f(x) + f(y) \neq f(x + y)] = \gamma < 2/9,$$

then there exists a linear function g such that $\Delta(f, g) < .1$

The consequence of the above theorem is the first non-trivial locally-checkable code which can be probed at $p = 3$ places such that words which are not uniquely decodable are rejected with probability at least $\gamma = 2/9 > 0$.

Corollary 9 *The Hadamard codes are $(3, 2/9)$-locally checkable.*

Aside: Before we go on to describe the local checkability properties of higher-degree codes, we digress for a moment to pay our tribute to an invaluable source for information on the testability of the Hadamard codes – Don Coppersmith. The Linearity Test Theorem in Blum, Luby and Rubinfeld [15] is more general and applies to the case of testing homomorphisms among groups. The proof presented by Blum, Luby and Rubinfeld [15] is different from, and simpler than, their original one and is attributed to Don Coppersmith. Coppersmith, in some unpublished notes, has a proof of the Linearity Test Theorem in its tightest form (including the bounds mentioned above) along with examples demonstrating the tightness of the theorem for the case when f maps from groups G to H. For the case, where we restrict our attention to functions arising from Hadamard Codes, i.e., functions mapping from $\mathcal{Z}_2^m$ to $\mathcal{Z}_2$, the analysis above turns out not to be tight. This is also shown by Coppersmith, in an unpublished work with Bellare and Sudan, where the bound on γ above is raised to $45/128$. Once again counterexamples are provided to demonstrate the tightness of the bounds achieved here. Perhaps even more significant than the tight analysis is the paradigm developed by Coppersmith for the proof of a local checkability property, which has become a standard in the works of Gemmell, Lipton, Rubinfeld, Sudan and Wigderson [23]; Rubinfeld and Sudan [33, 34]; Arora, Lund, Motwani, Sudan and Szegedy [2] and Friedl and Sudan [21].

We now turn our attention to the local checkability properties of low-degree codes, for larger degree. The task of testing such functions has been studied in the recent past in numerous papers (see [6, 7, 23, 17, 33, 34, 3, 2, 32, 21]). Of these a subset of the papers focuses on the task of testing the total-degree of polynomials [23, 33, 34, 2, 21] and a different subset focuses on individual degree testing [6, 7, 17, 3, 32]. The latter family does not directly provide us with the best possible locally-checkable codes, but nevertheless plays a crucial role in the analysis of the former family. In particular the work of Arora and Safra [3] (see also subsequent work by Polishchuk and Spielman [32]) is the prime reason for the improved analysis of the low-degree test in the work of Arora, Lund, Motwani, Sudan and Szegedy [2]. Giving details of the works in all these papers is beyond the scope of this article and we refer the reader to Arora [1] or Sudan [37] for further details. We state the theorem that culminates from these results.

Theorem 10 ([34, 3, 2, 21]) *For all $\epsilon > 0$, there exist $[k^{2+\epsilon}, k, 1/4]$ codes over alphabets of size $2^{\mathrm{polylog}\, k}$, that are $(2, 1/8 - \epsilon)$ locally checkable.*

The existence of such codes along with a recursive technique developed by Arora and Safra [3] is used by Arora, Lund Motwani, Sudan and Szegedy to construct good codes over a binary alphabet that is (p, γ) locally-checkable for $p < \infty$ and $\gamma > 0$. The recursive technique of Arora and Safra [3] deserves special mention. In a sense, it provides a proof-checking analog of the recursive construction used earlier to combine the Reed-Solomon codes with Hadamard codes.

5 Polynomials and complexity theory

We are now on the last leg of this journey through the results in complexity theory. A reader going through this writeup carefully will notice that the same set of papers get referenced in the different sections. This is due to the fact that each paper has involved some contributions in terms of local-checkability and some in terms of complexity theory. Thus in actuality these papers don't really separate the two aspects of the works from one another. There does exist one major exception to this rule - the work of Babai and Fortnow [5]. Their work extracts algebraic characterizations of $\sharp\mathrm{P}$ and PSPACE and uses that to explain the existence of efficient interactive proofs that exist for the two classes.

This work in turn motivated the search for a similar explanation of the PCP characterization of NP. Such a characterization was extracted eventually by this author [37] based on the works of Babai, Fortnow and Lund [6] and the subsequent work of Babai, Fortnow, Levin and Szegedy [7].

The characterization is based on the notion of a construction rule for a sequence of polynomials.

Definition 11 (polynomial construction rule) *Given an polynomial $g^{(0)}$, a construction rule for a sequence of degree d polynomials $g^{(1)}, \ldots, g^{(l)}$ is a set of rules $r_1, \ldots, r_l$, where r_i describes how to evaluate the polynomial $g^{(i)}$ at any point $\hat{x} \in F^m$ using oracles for the previously defined polynomials $g^{(0)}, \ldots, g^{(i-1)}$.*

The only constraint on the rule r_i is that its representation should be polynomially bounded in the input space size (i.e., $O(\mathrm{poly}(|F|^m))$). l is referred to as the length *of such a rule. The maximum number of oracle calls made by any rule to evaluate any input is called the* width *of the rule and is denoted w.*

Definition 12 *A language L is (l, w, m, d, a)* polynomial describable *if for all $x \in L$, a construction rule $(r_x^{(1)}, \ldots, r_x^{(l)}$ of length l and width w for degree d polynomials in m variables over a field F of size 2^a can be generated in polynomial time (in the length of x), with the following properties:*

- *If $x \in L$ then there exists a degree d polynomial $g^{(0)}$ such that the polynomial $g^{(l)}$ obtained by applying the construction rules $(r_x^{(1)}, \ldots, r_x^{(l)}$ is identically 0.*
- *If $x \notin L$ then for any polynomial $g^{(0)}$ the application of the construction rules will yield a non-zero polynomial.*

The following characterization is described in [37] and uses the works of Babai, Fortnow, Lund [6] and Babai, Fortnow, Levin and Szegedy [7].

Theorem 13 ([37]) *Every language in* NP *is* $(\log, \log, \frac{\log}{\mathrm{loglog}}, \mathrm{polylog}, \mathrm{polylog})$ *polynomial describable.*

The result of Babai, Fortnow, Levin and Szegedy $\mathrm{NP} \subset \mathrm{PCP}(\mathrm{polylog}, \mathrm{polylog})$ is shown by Sudan [37] to be an immediate consequence of this characterization and the local checkability properties of polynomials described in Section 4. Implicit in the work of Arora, Lund, Motwani, Sudan and Szegedy is also a different containment result for NP.

Theorem 14 ([2]) NP *is* $(2, 2, \mathrm{poly}, 2, 1)$ *polynomial describable.*

The theorem above is surprisingly low in its parameters - all except for the number of variables. The contrast between the two theorems above should remind the reader of the contrast between the Reed-Solomon codes and the Hadamard code. This similarity in the two cases is not coincidental and the reader is urged to look at the full papers for more details on the connections. The $\mathrm{NP} = \cup_k \mathrm{PCP}(k \log, C)$ result of Arora, Lund, Motwani, Sudan and Szegedy [2] is obtained by combining the two different ensuing PCP results by the recursive composition paradigm of Arora and Safra [3]. Once again details of this can be found in [2].

6 Conclusion

The aim of this hastily written article was to bring to light the different elements of algebra that play a role in the results on efficient proof verification. Since Theorem 3 was discovered, a significant amount or work has gone into making these proof systems efficient and large degree of success has been achieved. We conclude by describing this sequence of results:

- Bellare, Goldwasser, Lund and Russell [9] initiated the investigation of how small can the constant C be in Theorem 3. In a surprising result they show how this number can be reduced (from some unmentioned but huge constant in [2]) to 29 bits! The number has reduced, steadily but not dramatically, even since - with Feige and Kilian [18] achieving 24 bits, and in an unpublished work Bellare, Goldreich and Sudan achieving 16.87 bits on the average.

- The verifier created by Bellare, Goldwasser, Lund and Russell [9] and the subsequent works, require a proof to be blown by enormous (though only polynomially larger) factors. Polishchuk and Spielman [32] investigate the question of how small the proofs could be and show that proofs need not be increased by large factors - an increase from n to $n^{1+\epsilon}$ suffices for there to exist a verifier who probes the proof in only a constant number of bits.
- Unfortunately, the Polishchuk and Spielman [32] result increases the constant C back to the level used in [2]. Is it possible to reduce the number of probes and the proof size simultaneously? This question was considered by Friedl and Sudan [21] who show that with a slightly super-quadratic blowup the proof can be made robust enough that 165 bits probed into the proof suffice.

It remains open if we can ever get linear sized proofs with (small) constant probe complexity.

Acknowledgments

I'm grateful to Mihir Bellare for pointing out the numerous errors that appeared in an earlier version and for suggesting corrections that have improved the quality of this article significantly.

References

[1] S. ARORA. *Probabilistic Checking of Proofs and Hardness of Approximation Problems.* Ph. D. Thesis, Computer Science Division, University of California at Berkeley, 1994.

[2] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN AND M. SZEGEDY. Proof verification and intractability of approximation problems. *Proceedings of the Thirty Third Annual Symposium on the Foundations of Computer Science,* IEEE, 1992.

[3] S. ARORA AND S. SAFRA. Probabilistic checking of proofs: a new characterization of NP. *Proceedings of the Thirty Third Annual Symposium on the Foundations of Computer Science*, IEEE, 1992.

[4] L. BABAI. Transparent (holographic) proofs. This STACS not yet defined! .

[5] L. BABAI AND L. FORTNOW. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[6] L. BABAI, L. FORTNOW, AND C. LUND. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1 (1991), 3–40.

[7] L. BABAI, L. FORTNOW, L. LEVIN, AND M. SZEGEDY. Checking computations in polylogarithmic time. *Proceedings of the Twenty Third Annual Symposium on the Theory of Computing,* ACM, 1991.

[8] L. BABAI AND S. MORAN. Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[9] M. BELLARE, S. GOLDWASSER, C. LUND AND A. RUSSELL. Efficient probabilistically checkable proofs and applications to approximation. *Proceedings of the Twenty Fifth Annual Symposium on the Theory of Computing,* ACM, 1993. See also Errata sheet in *Proceedings of the Twenty Sixth Annual Symposium on the Theory of Computing,* ACM, 1994.

[10] M. Bellare and P. Rogaway. The complexity of approximating a nonlinear program. *Complexity of Numerical Optimization*, Ed. P.M. Pardalos, World Scientific (1993).

[11] M. Bellare and M. Sudan. Improved non-approximability results. *Proceedings of the Twenty Sixth Annual Symposium on the Theory of Computing,* ACM, 1994.

[12] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson. Multi-Prover interactive proofs: How to remove intractability assumptions. *Proceedings of the Twentieth Annual Symposium on the Theory of Computing,* ACM, 1988.

[13] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Information and Computation* **96**, 77–94 (1992).

[14] M. Blum and S. Kannan. Program correctness checking . . . and the design of programs that check their work. *Proceedings of the Twenty First Annual Symposium on the Theory of Computing,* ACM, 1989.

[15] M. Blum, M. Luby and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Proceedings of the Twenty Second Annual Symposium on the Theory of Computing,* ACM, 1990.

[16] R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, June 1978.

[17] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *Proceedings of the Thirty Second Annual Symposium on the Foundations of Computer Science,* IEEE, 1991.

[18] U. Feige and J. Kilian. Two prover protocols - low error at affordable rates. *Proceedings of the Twenty Sixth Annual Symposium on the Theory of Computing,* ACM, 1994.

[19] U. Feige and L. Lovasz. Two-prover one-round proof systems: Their power and their problems. *Proceedings of the Twenty Fourth Annual Symposium on the Theory of Computing*, ACM, 1992.

[20] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Proceedings of the Third Annual Conference on Structure in Complexity Theory*, IEEE, 1988.

[21] K. Friedl and M. Sudan. Some improvements to total degree tests. *Proceedings of the Third Israel Symposium on Theory and Computing Systems*, 1995.

[22] K. Friedl, Zs. Hátsági, and A. Shen. Low-degree tests. *Proceedings of the Fifth Symposium on Discrete Algorithms*, ACM, 1994.

[23] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. *Proceedings of the Twenty Third Annual Symposium on the Theory of Computing,* ACM, 1991.

[24] O. Goldreich. A taxonomy of proof systems. *SIGACT News*, Complexity Theory Column, Two part series appears in December 1993, and March 1994.

[25] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[26] D. JOHNSON. The tale of the second prover. *Journal of Algorithms*, 13:502–524, 1992. Part of *The NP-Completeness Column: An Ongoing Guide*.

[27] R. KARP. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Miller and Thatcher (eds.), Plenum Press, New York (1972).

[28] C. LUND, L. FORTNOW, H. KARLOFF, AND N. NISAN. Algebraic methods for interactive proof systems. *Proceedings of the Thirty First Annual Symposium on the Foundations of Computer Science*, IEEE, 1990.

[29] C. LUND AND M. YANNAKAKIS. On the hardness of approximating minimization problems. *Proceedings of the Twenty Fifth Annual Symposium on the Theory of Computing*, ACM, 1993.

[30] F. MACWILLIAMS AND N. SLOANE. *The Theory of Error-Correcting Codes*. North-Holland. 1981.

[31] C. PAPADIMITRIOU AND M. YANNAKAKIS. Optimization, approximation, and complexity classes. *Proceedings of the Twentieth Annual Symposium on the Theory of Computing*, ACM, 1988.

[32] A. POLISHCHUK AND D. SPIELMAN. Nearly-linear size holographic proofs. *Proceedings of the Twenty Sixth Annual Symposium on the Theory of Computing*, ACM, 1994.

[33] R. RUBINFELD AND M. SUDAN. Testing polynomial functions efficiently and over rational domains. *Proceedings of the Third Symposium on Discrete Algorithms*, ACM, 1994.

[34] R. RUBINFELD AND M. SUDAN. Robust characterizations of polynomials with applications to program testing. *Technical Report RC 19156, IBM Research Division*, T. J. Watson Research Center, Yorktown Heights, NY 10598, September 1993.

[35] A. SHAMIR. IP = PSPACE. *Proceedings of the Thirty First Annual Symposium on the Foundations of Computer Science*, IEEE, 1990.

[36] J. T. Schwartz. Fast probabilistic algorithms for verification of probabilistic identities. *Journal of the ACM*, v. 27, 701–717, 1980.

[37] M. SUDAN. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems.* Ph.D. Thesis, Computer Science Division, University of California at Berkeley. 1992.

[38] R. ZIPPEL. Probabilistic algorithms for sparse polynomials. *EUROSAM '79, Lecture Notes in Computer Science*, 72:216–226, 1979.