Short Locally Testable Codes and Proofs (Survey)

Oded Goldreich*
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

Preliminary version: July 2004

Abstract

We survey known results regarding locally testable codes and locally testable proofs (known as PCPs), with emphasis on the length of these constructs. Locally testability refers to approximately testing large objects based on a very small number of probes, each retrieving a single bit in the representation of the object. This yields super-fast approximate-testing of the corresponding property (i.e., be a codeword or a valid proof). We also review the related concept of local decodable codes.

The currently best results regarding locally testable codes and proofs demonstrate a trade-off between the number of probes and the length of the code or proof (relative to the information that it encodes). Actually, the length is always "nearly linear", and the trade-off is between number of probes and the "level of near-linearity". Needless to say, it is not clear whether this trade-off is inherent.

The survey consists of two independent (i.e., self-contained) parts that cover the same material at different levels of rigor and detail. Still, in spite of the repetitions, there may be a benefit in reading both parts.

Keywords: Error Correcting Codes, Probabilistically Checkable Proofs (PCP), Locally Testable Codes, Locally Decodable Codes, Property Testing, Self-Correction, Low-Degree Tests, Derandomization, and Private Information Retrieval.

*Written while being a fellow at the Radcliffe Institute for Advanced Study of Harvard University.

Contents

Part I
A high-level overview

The title of this survey refers to two types of objects (i.e., codes and proofs) and two adjectives: *local testability* and being *short*. A clarification of these terms is in place.

Codes, proofs and their length. Codes are sets of strings (of equal length), typically, having a large pairwise distance. Equivalently, codes are viewed as mappings from short (k-bit) strings to longer (n-bit) strings, called codewords, such that the codewords are distant from one another. We will focus on *codes with relative constant distance*; that is, every two n-bit codewords are at distance $\Omega(n)$ apart. The length of the code is measured in terms of the length of the pre-image (i.e., we are interested in the growth of n as a function of k). Turning to proofs, these are defined with respect to a verification procedure for assertions of a certain length, and their length is measured in terms of the length of the assertion. The verification procedure must satisfy the natural completeness and soundness properties: For valid assertions there should be strings, called proofs, that are accepted (in conjunction with the assertion) by the verification procedures, whereas for false assertions no such strings may exist. The reader may envision proof systems for the set of satisfiable propositional formulae (i.e., assertions of satisfiability of given formulae).

Local testability. By local testability we mean that the object can be tested for the natural property (i.e., being a codeword or a valid proof) using a small (typically constant) number of probes, each recovering individual bits in a standard representation of the object. Thus, local testability allows for super-fast testing of the corresponding fundamental objects. The tests are probabilistic and hence the result is correct only with high probability.[1] Furthermore, correctness refers to a *relaxed notion of deciding* (which was formulated, in general terms, in the context of property testing [41, 30]): It is required that valid objects be accepted with high probability, whereas objects that are "far" from being valid should be rejected with high probability. Specifically, in case of codes, codewords should be accepted (with high probability), whereas strings that are "far" from the code should be rejected (with high probability). In case of proofs, valid proofs (which exist for correct assertions) should be accepted (with high probability), whereas strings that are "far" from being valid proofs (and, in particular, all strings in case no valid proofs exist) should be rejected (with high probability).[2]

Our notion of locally testable proofs is very related to the notion of a PCP (i.e., probabilistically checkable proof)[3], and we will ignore the difference in the sequel. The difference is that in the definition of locally testable proofs we required rejection of strings that are far from any valid proof, also in the case that valid proofs exists (i.e., the assertion is valid). In contrast, the standard rejection criteria of PCPs refers only to false assertions. Still, all known PCP constructions actually satisfy the stronger definition.[4]

[1]Indeed, it is easy to see that deterministic tests will perform very poorly, and the same holds with respect to probabilistic tests that make no error.

[2]Indeed, in the case the assertion is false, there exist no valid proofs. In this case all strings are defined to be far from a valid proof.

[3]Needless to say, the new term "locally testable proof" was introduced to match the term "locally testable codes". In retrospect, "locally testable proofs" seems a more fitting term than "probabilistically checkable proofs", because it stresses the positive aspect (of locality) rather than the negative aspect (of being probabilistic). The latter perspective has been frequently advocated by Leonid Levin.

[4]In some cases this holds only under a weighted version of the Hamming distance, rather under the standard

The very possibility of local testability. Indeed, local testability of either codes or proofs is quite challenging, regardless of the issue of length:

- For codes, the simplest example of a locally testable code (of constant relative distance) is the Hadamard code and testing it reduces to linearity testing. However, the analysis of the natural linearity tester (of Blum, Luby and Rubinfeld [19]) turned out to be highly complex (cf. [19, 6, 25, 12, 13, 10]).
- For proofs, the simplest example of a locally testable proof is the "inner verifier" of the PCP construction of Arora, Lund, Motwani, Sudan and Szegedy [4], which in turn is based on the Hadamard code.

In both cases, the constructed object has exponential length in terms of the relevant parameter (i.e., the amount of information being encoded in the code or the length of the assertion being proved).

Local testability at a polynomial blow-up. Achieving local testability by codes and proofs that have polynomial length turns out to be even more challenging.

- In the case of codes, a direct interpretation of *low-degree tests* (cf. [6, 7, 29, 41, 28]), proposed in [28, 41], yields a locally testable code of quadratic length over a *sufficiently large alphabet*. Similar (and actually better) results for *binary* codes required additional ideas, and have appeared only later (cf. [32]).
- The case of proofs is far more complex: Achieving locally testable proof of polynomial length is essentially the contents of the celebrated PCP Theorem of Arora, Lund, Motwani, Safra, Sudan and Szegedy [5, 4].

We focus on even *shorter* codes and proofs; specifically, codes and proofs of *nearly linear length*. The latter term has been given quite different interpretations, and we start by sorting these out.

Types of nearly linear functions: A few common interpretations of this term are listed below (going from the most liberal to the most strict one).

T1-nearly linear: A very liberal notion, at the verge of an abuse of the term, refers to a sequence of functions $f_\epsilon : \mathsf{N} \to \mathsf{N}$ such that, for every $\epsilon > 0$, it holds that $f_\epsilon(n) \leq n^{1+\epsilon}$. That is, each function is actually of the form $n \mapsto n^c$, for some constant $c > 1$, but the sequence as a whole can be viewed as approaching linearity.

T2-nearly linear: A more reasonable notion of nearly linear functions refers to individual functions f such that $f(n) = n^{1+o(1)}$. Specifically, for some function $\epsilon : \mathsf{N} \to [0,1]$ that goes to zero, it holds that $f(n) \leq n^{1+\epsilon(n)}$. Common sub-types include the case that $\epsilon(n) = 1/\log\log n$, the case that $\epsilon(n) = 1/(\log n)^c$ for some $c \in (0,1)$, and the case that $\epsilon(n) = \exp((\log\log\log n)^c)/\log n$ for some $c \in (0,1)$. Indeed, the case in which $\epsilon(n) = O(\log\log n)/\log n$ (or so) deserves a special category.

Hamming distance. Alternatively, these constructions can be easily modified to work under the standard Hamming distance.

T3-nearly linear: The strongest notion interprets near-linearity as linearity up to a poly-logarithmic (or quasi-poly-logarithmic) factor. In the former case $f(n) \leq \text{poly}(\log n) \cdot n$, which corresponds to the aforementioned case of $f(n) \leq n^{1+\epsilon(n)}$ with $\epsilon(n) = O(\log\log n)/\log n$, whereas the latter case corresponds to $\epsilon(n) = \text{poly}(\log\log n)/\log n$ (i.e., in which case $f(n) = (\log n)^{\text{poly}(\log\log n)} \cdot n$).

Using the above notation, we summarize the state of the art with respect to local testability of codes and proofs.

Local testability with nearly linear length: The ultimate goal may be to obtain locally testable codes and proofs that are T3-nearly linear (i.e., nearly linear in the sense of Type T3). We conjecture that locally testable codes and proofs of (strictly) linear length cannot be achieved. Currently, locally testable codes and proofs of nearly linear length are known when nearly linear is interpreted as Type T2 (i.e., T2-nearly linear).

Theorem 1 (Ben-Sasson, Goldreich, Harsha, Sudan and Vadhan [15]): *There exist locally testable codes and proofs of length $f(n) \leq n^{1+\epsilon(n)}$, where $\epsilon(n) = 1/(\log n)^{0.99}$. Actually, for every constant $c \in (0,1)$, one can achieve length $f(n) \leq n^{1+\epsilon(n)}$, where $\epsilon(n) = 1/(\log n)^c$.*

Open Problem 2 *Do there exist locally testable codes and proofs of length $f(n) \leq \text{poly}(\log n) \cdot n$?*

In the rest of this part, we motivate the study of short locally testable objects, comment on the relation between such codes and proofs, and discuss a somewhat related coding problem.

Motivation to the study of short locally testable codes and proofs

Local testability offers an extremely strong notion of efficient testing: The tester makes only a constant number of bit probes, and determining the probed locations (as well as the final decision) is typically done in time that is poly-logarithmic in the length of the probed object.

The length of an error-correcting code is widely recognized as one of the two most fundamental parameters of the code (the second one being its distance). In particular, the length of the code is of major importance in applications, because it determines the overhead involved in encoding information.

The same considerations apply also to proofs. However, in the case of proofs, this obvious point was blurred by the indirect, unexpected and highly influential applications of locally testable proofs (known as PCPs) to the theory of approximation algorithms. In our view, the significance of locally testable proofs (i.e., PCPs) extends far beyond their applicability to deriving non-approximability results. The mere fact that proofs can be transformed into a format that supports super-fast probabilistic verification is remarkable. From this perspective, the question of how much redundancy is introduced by such a transformation is a fundamental one. Furthermore, locally testable proofs (i.e., PCPs) have been used not only to derive non-approximability results but also for obtaining positive results (e.g., CS-proofs [36, 39] and their applications [8, 21]), and the length of the PCP affects the complexity of those applications.

In any case, the length of PCPs is also relevant to non-approximability results; specifically, it affects their *tightness with respect to the running time*. For example, suppose (exact) SAT has complexity $2^{\Omega(n)}$. The original PCP Theorem [5, 4] only implies that approximating MaxSAT requires time 2^{n^α}, for some (small) $\alpha > 0$. The work of [40] makes α arbitrarily close to 1, whereas the results of [32, 18] further improve the lower-bound to $2^{n^{1-o(1)}}$.[5]

On the relation between locally testable codes and proofs

Locally testable codes seem related to locally testable proofs (PCPs). In fact, the use of codes with related "local testability" features is implicit in known PCP constructions. Furthermore, the known constructions of locally testable proofs (PCPs) provides a transformation of *standard proofs* (for say SAT) to *locally testable proofs* (i.e., PCP-oracles), such that transformed strings are accepted with probability one by the PCP verifier. Moreover, starting from different standard proofs, one obtains locally testable proofs that are far apart, and hence constitute a good code. It is tempting to think that the PCP verifier yields a codeword tester, but this is not really the case. Note that our definition of a locally testable proof requires rejection of strings that are far from any valid proof, but it is not clear that the only valid proofs (w.r.t the constructed PCP verifier) are those that are obtained by the aforementioned transformation of standard proofs to locally testable ones.[6] In fact, the standard PCP constructions accept also valid proofs that are not in the range of the corresponding transformation.

In spite of the above, locally testable codes and proofs are related, and the feeling is that locally testable codes are the combinatorial counterparts of locally testable proofs (PCPs), which are complexity theoretic in nature. From that perspective, one should expect (or hope) that it would be easier to construct locally testable codes than it is to construct PCPs. This feeling was among the main motivations of Goldreich and Sudan, and indeed their first result was along this vein: They showed a relatively simple construction (i.e., simple in comparison to PCP constructions) of a locally testable code of T1-nearly linear length [32, Sec. 3]. Unfortunately, their stronger result, providing a locally testable code of T2-nearly linear length is obtained by constructing and using a T2-nearly linear locally testable proof (i.e., PCP). Subsequent works [18, 15] have followed this route, and only the recent work of Ben-Sasson and Sudan [17] (which achieves a more relaxed notion of local testability) reversed the course to the "right one": First codes are constructed, and next they are used towards the construction of proofs (rather than the other way around).

Locally Decodable Codes

Locally *decodable* codes are in some sense complimentary to local *testable* codes. Here, one is given a slightly corrupted codeword (i.e., a string close to some unique codeword), and is required to recover individual bits of the encoded information based on a constant number of probes (per recovered bit). That is, a code is said to be locally decodable if whenever relatively few location are corrupted, the decoder is able to recover each information-bit, with high probability, based on a constant number of probes to the (corrupted) codeword.

The best known locally decodable codes are of sub-exponential length. Specifically, k information bits can be encoded by codewords of length $n = \exp(k^{O(\log\log q)/q\log q})$ that are locally decodable using q bit-probes (cf. [9]). It is conjectured that, for every q there exists an $\epsilon > 0$, such that locally decodability based on q queries (i.e., probes) requires codewords of length $n > \exp(k^\epsilon)$. The problem is related to the construction of (information theoretic secure) Private Information Retrieval schemes, introduced in [22].

[5] A caveat: it is currently not known whether these improved lower-bounds can be achieved simultaneously with optimal approximation ratios, but the hope is that this can eventually be done.

[6] Let alone that the standard definition of PCP refers only to the case of false assertions, in which case all strings are far from a valid proof (which does not exist).

A natural relaxation of the definition of locally decodable codes requires that, whenever few location are corrupted, the decoder should be able to recover most of the individual information-bits (based on a constant number of queries) and for the rest of the locations, the decoder may output a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but on a few bit-locations it is allowed to sometimes say "`don't know`". This relaxed notion of local decodability can be supported by codes that have T1-nearly linear length (cf. [15]).

An obvious open problem is to separate locally decodable codes from relaxed locally decodable codes. This may follow by either improving the $\Omega(k^{1+\frac{1}{q-1}})$ lower bound on the length of q-query locally decodable codes (of [34]), or by providing relaxed locally decodable codes of T2-nearly linear length.

Part II
A more detailed and rigorous account

In this part we provide a general treatment of local testability. In contrast to Part I, here we allow the tester to use a number of queries that is a (typically small) predetermined function of the length parameter, rather than insisting on a constant number of queries. The latter special case is indeed an important one.

1 Introduction

Codes (i.e., error correcting codes) and proofs (i.e., automatically verifiable proofs) are fundamental to computer science as well as to related disciplines such as mathematics and computer engineering. Redundancy is inherent to error-correcting codes, whereas testing validity is inherent to proofs. In this survey we also consider less traditional combinations such as testing validity of codewords and the use of proofs that contain redundancy. The reader may wonder why we explore these non-traditional possibilities, and the answer is that they offer various advantages (as will be elaborated next).

Testing the validity of codewords is natural in settings in which one may want to take an action in case the codeword is corrupted. For example, when storing data in an error correcting format, one may want to recover the data and re-encode it whenever one finds that the current encoding is corrupted. Doing so may allow to maintain the data integrity over eternity, when encoded bits do get corrupted in the course of time. Of course, one can use the error-correcting decoding procedure associated with the code in order to check whether the current encoding is corrupted, but the question is whether one can check (or just approximately check) this property *much faster*.

Loosely speaking, locally testable codes are error correcting codes that allow for a super-fast testing of whether or not a give string is a valid codeword. In particular, the tester works in sub-linear time and reads very few of the bits of the tested object. Needless to say, the answer provided by such a tester can only be approximately correct, but this would suffice in many applications (including the one sketched above).

Similarly, locally testable proofs are proofs that allow for a super-fast probabilistic verification. Again, the tester works in sub-linear time and reads very few of the bits of the tested object. The tester's (aka verifier's) verdict is only correct with high probability, but this may suffice for many applications. In particular, it suffices in applications where proofs refer to the correctness of a specific computation of practical interest (rather than referring to Fermat's Theorem). Lastly, we comment that such *locally testable proofs must be redundant* (or else there would be no chance for verifying them based on inspecting only a small portion of them).

Our focus is on relatively *short* locally testable codes and proofs, which is not surprising in view of the fact that *we envision such objects being actually used in practice*. Of course, we do not mean to suggest that one may use in practice any of the constructions surveyed here (especially not the ones that provide the stronger bounds). We rather argue that this direction of research may find applications in practice. Furthermore, it may even be the case that some of the current concepts and techniques may lead to such applications.

Organization: In Section 2 we provide a quite comprehensive definitional treatment of locally testable codes and proofs, while relating these to PCPs, PCPs of proximity, and property testing. In Section 3, we survey the main results regarding locally testable codes and proofs as well as many of the underlying ideas. In Section 4 we consider locally decodable codes, which are somewhat complementary to locally testable codes.

2 Definitions

Local testability is formulated by considering oracle machines. That is, the tester is an oracle machine, and the object that it tests is viewed as an oracle. For simplicity, we confine ourselves to *non-adaptive* probabilistic oracle machines; that is, machines that determine their queries based on their explicit input (which in case of codes is merely a length parameter) and their internal coin tosses (but not depending on previous oracle answers). When taking about oracle access to a string $w \in \{0,1\}^n$ we viewed w as a function $w : \{1,...,n\} \to \{0,1\}$.

2.1 Codeword testers

We consider codes mapping sequences of k (input) bits into sequences of $n \geq k$ (output) bits. Such a generic code is denoted by $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, and the elements of $\{\mathrm{C}(x) : x \in \{0,1\}^k\} \subseteq \{0,1\}^n$ are called codewords (of C).

The distance of a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ is the minimum (Hamming) distance between its codewords; that is, $\min_{x \neq y}\{\Delta(\mathrm{C}(x), \mathrm{C}(y))\}$, where $\Delta(u,v)$ denotes the number of bit-locations on which u and v differ. Throughout this work, *we focus on codes of linear distance*; that is, codes $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ of distance $\Omega(n)$.

The distance of $w \in \{0,1\}^n$ from a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, denoted $\Delta_{\mathrm{C}}(w)$, is the minimum distance between w and the codewords; that is, $\Delta_{\mathrm{C}}(w) \stackrel{\rm def}{=} \min_x\{\Delta(w, \mathrm{C}(x))\}$. For $\delta \in [0,1]$, the n-bit long strings u and v are said to be δ-far (resp., δ-close) if $\Delta(u,v) > \delta \cdot n$ (resp., $\Delta(u,v) \leq \delta \cdot n$). Similarly, w is δ-far from C (resp., δ-close to C) if $\Delta_{\mathrm{C}}(w) > \delta \cdot n$ (resp., $\Delta_{\mathrm{C}}(w) \leq \delta \cdot n$).

Definition 2.1 *Let* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *be a code of distance* d*, and let* $q \in \mathsf{N}$ *and* $\delta \in (0,1)$*. A* q-local (codeword) δ-tester for C *is a probabilistic* (non-adaptive) *oracle machine* M *that makes at most* q *queries and satisfies the following two conditions:*

Accepting codewords (aka completeness): *For any* $x \in \{0,1\}^k$*, given oracle access to* $w = \mathrm{C}(x)$*, machine* M *accepts with probability 1. That is,* $\Pr[M^{\mathrm{C}(x)}(1^k) = 1] = 1$*, for any* $x \in \{0,1\}^k$*.*

Rejection of non-codeword (aka soundness): *For any* $w \in \{0,1\}^n$ *that is* δ*-far from* C*, given oracle access to* w*, machine* M *rejects with probability at least* $1/2$*. That is,* $\Pr[M^w(1^k) = 1] \leq 1/2$*, for any* $w \in \{0,1\}^n$ *that is* δ*-far from* C*.*

We call q *the* query complexity *of* M*, and* δ *the* proximity parameter*.*

The above definition is interesting only in case δn is smaller than the covering radius of C (i.e., the smallest r such that for every $w \in \{0,1\}^n$ it holds that $\Delta_{\mathrm{C}}(w) \leq r$). Clearly, $r \geq d/2$, and so the definition is certainly interesting in the case that $\delta < d/2n$, and indeed we will focus on this case. On the other hand, observe that $q = \Omega(1/\delta)$ must hold, which means that we focus on the case that $d/n = \Omega(1/q)$.

We next consider families of codes $\mathrm{C} = \{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k \in K}$, where $n, d : \mathsf{N} \to \mathsf{N}$ and $K \subseteq \mathsf{N}$, such that C_k has distance $d(k)$. In accordance with the above, we care most of the case that $\delta(k) < d(k)/2n(k)$. Furthermore, seeking constant query complexity, we focus on the case $d = \Omega(n)$.

Definition 2.2 *For functions $n, d : \mathsf{N} \to \mathsf{N}$, let $\mathrm{C} = \{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k\in K}$ such that C_k is a code of distance $d(k)$. For function $q : \mathsf{N} \to \mathsf{N}$ and $\delta : \mathsf{N} \to (0,1)$, we say that M is* q-local (codeword) δ-tester *for $\mathrm{C} = \{\mathrm{C}_k\}_{k\in K}$ if, for every $k \in K$, machine M is a $q(k)$-local $\delta(k)$-tester for C_k. Again, q is called the* query complexity *of M, and δ the* proximity parameter.

Recall that being particularly interested in constant query complexity (and recalling that $d(k)/n(k) \geq 2\delta(k) = \Omega(1/q(k))$), we focus on the case that $d = \Omega(n)$ and constant $\delta < d/2n$. In this case, we may consider a stronger definition.

Definition 2.3 *Let n, d and C be as in Definition 2.2 and suppose that $d = \Omega(n)$. We say that C is* locally testable *if for every constant $\delta > 0$ there exists a constant q and a probabilistic polynomial-time oracle machine M such that M is a q-local δ-tester for C.*

We will be concerned of the growth rate of n as a function of k, for locally testable codes $\mathrm{C} = \{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k\in K}$ of distance $d = \Omega(n)$. More generally, for $d = \Omega(n)$, we will be interested in the trade-off between n, the proximity parameter δ, and the query complexity q.

2.2 Proof testers

We start by recalling the standard definition of PCP.

Definition 2.4 *A* probabilistically checkable proof (PCP) system *for a set S is a probabilistic* (non-adaptive) *polynomial-time oracle machine* (called verifier), *denoted V, satisfying*

Completeness: *For every $x \in S$ there exists an oracle π_x such that V, on input x and access to oracle π_x, always accepts x; that is, $\Pr[V^{\pi_x}(x)=1] = 1$.*

Soundness: *For every $x \notin S$ and every oracle π, machine V, on input x and access to oracle π, rejects x with probability at least $\frac{1}{2}$; that is, $\Pr[M^{\pi}(x)=1] \leq 1/2$,*

Let $Q_x(r)$ denote the set of oracle positions inspected by V on input x and random-tape $r \in \{0,1\}^{\mathrm{poly}(|x|)}$. The query complexity *of V is defined as $q(n) \stackrel{\rm def}{=} \max_{x\in\{0,1\}^n, r\in\{0,1\}^{\mathrm{poly}(n)}}\{|Q_x(r)|\}$. The* proof complexity *of V is defined as $p(n) \stackrel{\rm def}{=} \max_{x\in\{0,1\}^n}\{|\cup_{r\in\{0,1\}^{\mathrm{poly}(n)}} Q_x(r)|\}$.*

Note that in the case that the verifier V uses a logarithmic number of coin tosses, its proof complexity is polynomial. In general, the proof complexity is upper-bounded by $2^r \cdot q$, where r (resp., q) is the randomness (resp., query) complexity of the proof tester. Thus, the trade-off between the query complexity and the proof complexity is typically captured by the trade-off between the query complexity and the randomness complexity. Furthermore, focusing on the randomness complexity allows for better bounds when composing proofs (cf. §3.2.2).

All known PCP constructions can be easily modified such that the oracle locations accessed by V are a prefix of the oracle (i.e., $\cup_{r\in\{0,1\}^{\mathrm{poly}(|x|)}} Q_x(r) \subseteq \{1, ..., p(|x|)\}$, for every x).[7] (For simplicity, the reader may assume that this is the case throughout the rest of this exposition.) More importantly, all known PCP constructions can be easily modified to satisfy the following definition, which is closer in spirit to the definition of locally testable codes.

Definition 2.5 *For function $q : \mathsf{N} \to \mathsf{N}$ and $\delta : \mathsf{N} \to (0,1)$, we say that a PCP system V for a set S is a* q-locally δ-testable proof system *if it has query complexity q and satisfies the following condition*

[7] In fact, for every $x \in \{0,1\}^n$, it holds that $\cup_{r\in\{0,1\}^{\mathrm{poly}(n)}} Q_x(r) = \{1, ..., p(n)\}$.

Rejecting invalid proofs: *For every* $x \in \{0,1\}^*$ *and every oracle* π *that is* δ*-far from* $\Pi_x \stackrel{\rm def}{=} \{w : \Pr[V^w(x)=1] = 1\}$, *machine* V, *on input* x *and access to oracle* π, *rejects* x *with probability at least* $\frac{1}{2}$.[8]

The proof complexity of V *is defined as in Definition 2.4.*

Note that Definition 2.5 uses the tester V itself in order to define the set (denoted Π_x) of valid proofs (for $x \in S$). That is, V is used both to define the set of valid proofs and to test for the proximity of a given oracle to this set. A more general definition (presented next), refers to an arbitrary proof system, and lets Π_x equal the set of valid proofs (in that system) for $x \in S$. Obviously, it must hold that $\Pi_x \neq \emptyset$ if an only if $x \in S$. Typically, one also requires the existence of a polynomial-time procedure that, on input a pair (x, π), determines whether or not $\pi \in \Pi_x$.[9] For simplicity we assume that, for some function $p : \mathsf{N} \to \mathsf{N}$ and every $x \in \{0,1\}^*$, it holds that $\Pi_x \subseteq \{0,1\}^{p(|x|)}$. The resulting definition follows.

Definition 2.6 *Suppose that, for some function* $p : \mathsf{N} \to \mathsf{N}$ *and every* $x \in \{0,1\}^*$, *it holds that* $\Pi_x \subseteq \{0,1\}^{p(|x|)}$. *For functions* $q : \mathsf{N} \to \mathsf{N}$ *and* $\delta : \mathsf{N} \to (0,1)$, *we say that a probabilistic* (non-adaptive) *polynomial-time oracle machine* V *is a* q-locally δ-tester for the proof system $\{\Pi_x\}_{x\in\{0,1\}^*}$ *if* V *has query complexity* q *and satisfies the following conditions*

Technical condition: *On input* x, *machine* V *issues queries in* $\{1, ..., p(|x|)\}$.

Accepting valid proofs: *For every* $x \in \{0,1\}^*$ *and every oracle* $\pi \in \Pi_x$, *machine* V, *on input* x *and access to oracle* π, *accepts* x *with probability 1.*

Rejecting invalid proofs: *For every* $x \in \{0,1\}^*$ *and every oracle* π *that is* δ*-far from* Π_x, *machine* V, *on input* x *and access to oracle* π, *rejects* x *with probability at least* $\frac{1}{2}$.

The proof complexity *of* V *is defined as* p,[10] *and* δ *is called the* proximity parameter. *In such a case, we say that* $\Pi = \{\Pi_x\}_{x\in\{0,1\}^*}$ *is* q-locally δ-testable, *and that* $S = \{x \in \{0,1\}^* : \Pi_x \neq \emptyset\}$ *has* q-locally δ-testable proofs of length p.
We say that Π *is* locally testable *if for every constant* $\delta > 0$ *there exists a constant* q *such that* Π *is* q*-locally* δ*-testable. In such a case, we say that* S *has* locally testable proofs of length p.

2.3 Discussion

We first comment about a few definitional choices made above. Firstly, we chose to present testers that always accept valid objects (i.e., accept valid codewords (resp., valid proofs) with probability 1). This is more appealing than allowing two-sided error, but the latter weaker notion is meaningful

[8]The above definition relies on two natural conventions:

1. All strings in Π_x are of the same length, which equals $|\cup_{r\in\{0,1\}^{\mathrm{poly}(n)}} Q_x(r)|$, where $Q_x(r)$ is as in Definition 2.4. Furthermore, we consider only π's of this length.
2. If $\Pi_x = \emptyset$ then every π is considered δ-far from Π_x.

[9]We comment that in the case that the verifier V uses a logarithmic number of coin tosses, its proof complexity is polynomial (and so the "effective length" of the strings in Π_x must be polynomial in $|x|$). Furthermore, if in addition it holds that $\Pi_x = \{w : \Pr[V^w(x)=1] = 1\}$, then (scanning all possible coin tosses of) V yields a polynomial-time procedure for determining whether a given pair (x, π) satisfies $\pi \in \Pi_x$.

[10]Note that by the technical condition, the current definition of the proof complexity of V is lower-bounded by the definition used in Definition 2.4.

too. A second choice was to fix the error probability (i.e., probability of accepting far from valid objects), rather than introducing yet another parameter. Needless to say, the error probability can be reduced by sequential applications of the tester.

In the rest of this section, we consider an array of definitional issues. First, we consider two natural strengthenings of the definition of local testability (cf. §2.3.1). We next we discuss the relation of local testability to property testing (cf. §2.3.2), and the relation of locality testable proofs to PCP of proximity (as defined in [15], cf. §2.3.3). Finally, we discuss the relation between local testable codes and proofs (cf. §2.3.4), and the motivation to the study of *short* local testable codes and proofs (cf. §2.3.5). (The text regarding the last issue is almost identical to a corresponding text that appears in Part I.)

2.3.1 Stronger definitions

The definitions of testers presented so far, allow for the construction of a different tester for each relevant value of the proximity parameter. However, whenever such testers are actually constructed, they tend to be "uniform" over all relevant values of the proximity parameter. Thus, it is natural to present a single tester for all relevant values of the proximity parameter, provide this tester with the said parameter, allow it to behave accordingly, and measure its query complexity as a function of that parameter. For example, we may strengthen Definition 2.3, by requiring the existence of a function $q:(0,1)\to\mathsf{N}$ and an oracle machine M such that, for every constant $\delta>0$, all (sufficiently large) k and all $w\in\{0,1\}^{n(k)}$, the following conditions hold:

1. On input $(1^k,\delta)$, machine M makes $q(\delta)$ queries.
2. If w is a codeword of C then $\Pr[M^w(1^k,\delta)=1]=1$.
3. If w is δ-far from $\{\mathrm{C}(x):x\in\{0,1\}^k\}$ then $\Pr[M^w(1^k,\delta)=1]\leq 1/2$.

An analogous strengthening applies to Definition 2.6. A special case of interest is when $q(\delta)=O(1/\delta)$. In this case, it makes sense to ask whether or not an even stronger "uniformity" condition may hold. Like in Definitions 2.1 and 2.2 (resp., Definitions 2.5 and 2.6), the tester M is not given the proximity parameter (and so its query complexity cannot depend on it), but we only require it to reject with probability proportional to the distance of the oracle from the relevant set. For example, we may strengthen Definition 2.3, by requiring the existence of an oracle machine M and a *constant* q such that, for every constant $\delta>0$, every (sufficiently large) k and $w\in\{0,1\}^{n(k)}$, the following conditions hold:

1. On input 1^k, machine M makes q queries.
2. If w is a codeword of C then $\Pr[M^w(1^k,\delta)=1]=1$.
3. If w is δ-far from $\{\mathrm{C}(x):x\in\{0,1\}^k\}$ then $\Pr[M^w(1^k,\delta)=1]<1-O(\delta)$.

2.3.2 Relation to Property Testing

Locally testable codes (and their corresponding testers) are essentially special cases of property testing, as defined in [41, 30]. Specifically, the property being tested is membership in a predetermined code. The only difference between the definitions presented in Section 2.1 and the formulation that is standard in the property testing literature is that in the latter the tester is given the proximity parameter as input and determines its behavior (and in particular the number of queries) accordingly. This difference is eliminated in §2.3.1. We note, however, that most of the property testing literature is concerned with "natural" objects (e.g., graphs, sets of points, functions) presented in a "natural" form rather than with object designed artificially to withstand errors (i.e., codewords of error correcting codes).

Our general formulation of proof testing (i.e., Definition 2.6) can be viewed as a generalization of property testing. That is, we view the set Π_x as a set of objects having a certain x-dependent property (rather than as a set of valid proofs for some property of x). In other words, Definition 2.6 allows to consider properties that are parameterized by auxiliary information (i.e., x), whereas traditional property testing may be viewed as referring to the case that x only determines the length of strings in Π_x (e.g., $\Pi_x = \emptyset$ for every $x \notin \{1\}^*$ or, equivalently, $\Pi_x = \Pi_y$ for every $|x| = |y|$).[11]

2.3.3 Relation to PCPs of Proximity

Our definition of a locally testable proof is related but different from the definition of a PCP of proximity (appearing in [15]).[12] We start by reviewing the definition of PCP of proximity.

Definition 2.7 *A* PCP of proximity for a set S with proximity parameter δ *is a probabilistic* (non-adaptive) *polynomial-time oracle machine, denoted V, satisfying*

Completeness: *For every $x \in S$ there exists a string π_x such that V always accepts when given access to the oracle (x, π_x); that is,* $\Pr[V^{x,\pi_x}(1^{|x|})=1] = 1$.

Soundness: *For every x that is δ-far from $S \cap \{0,1\}^{|x|}$ and for every string π, machine V rejects with probability at least $\frac{1}{2}$ when given access to the oracle (x, π); that is,* $\Pr[M^{x,\pi}(1^{|x|})=1] \leq 1/2$.

The query complexity of V is defined as in case of PCP, but here also queries to the x-part are counted.

The oracle (x, π) is actually a concatenation of two oracles: the input-oracle x (which replaces an explicitly given input in the definitions of PCPs and locally testable proofs), and a proof-oracle π (exactly as in the prior definitions). Note that Definition 2.7 refers to the distance of the input-oracle to S, whereas locally testable proofs refer to the distance of the proof-oracle from the set Π_x of valid proofs of membership of $x \in S$.

Still, PCPs of proximity can be defined within the framework of locally testable codes. Specifically, consider an extension of Definition 2.6, where (relative) distances are measured according to a weighted Hamming distance; that is, for a weight function $\omega : \{1, ..., n\} \to [0, 1]$ and $u, v \in \{0,1\}^n$, we let $\delta_\omega(u, v) = \sum_{i=1}^{n} \omega(i) \cdot \Delta(u_i, v_i)$. (Indeed, the standard notion of relative distance between $u, v \in \{0,1\}^n$ is obtained by $\delta_\omega(u, v)$ when using the uniform weighting function (i.e., $\omega(i) = 1/n$ for every $i \in \{1, ..., n\}$).) Now, Definition 2.7 can be viewed as a special case of (the extended) Definition 2.6 when applied to the (rather artificial) set of proofs $\Pi_{1^n} = \{(x, \pi) : x \in S \cap \{0,1\}^n \wedge \pi \in \Pi'_x\}$, where $\Pi'_x = \{\pi : \Pr[V^{x,\pi}(1^{|x|}) = 1] = 1\}$, by using the weighted Hamming distance δ_ω for ω that is uniform on the input-part of the oracle; that is, for $(x, \pi), (x', \pi') \in \{0,1\}^{n+p}$, we use $\delta_\omega((x, \pi), (x', \pi')) \stackrel{\text{def}}{=} \Delta(x, x')/n$, which corresponds to $\omega(i) = 1/n$ if $i \in \{1, ..., n\}$ and $\omega(i) = 0$

[11] In fact, in the context of property testing, the length of the oracle must always be given to the tester (although some sources neglect to state this fact).

[12] We mention that PCP of proximity are almost identical to Assignment Testers, defined independently by Dinur and Reingold [23]. Both notions are (important) special cases of the general definition of a "PCP spot-checker" formulated before in [24].

otherwise. Alternatively, weights can be approximately replaced by repetitions (provided that the tester checks the consistency of the repetitions).[13]

We mention that PCPs of proximity (of constant query complexity) yield a simple way of obtaining locally testable codes. More generally, we can combine any code C_0 with any PCP of proximity V, and obtain a q-locally testable code with distance essentially determined by C_0 and rate determined by V, where q is the query complexity of V. Specifically, x will be encoded by appending $c = \mathrm{C}_0(x)$ by a proof that c is a codeword of C_0, and distances will be determined by the weighted Hamming distance that assigns uniform weights to the first part of the new code. As in the previous paragraph, these weights can be implemented by making suitable repetitions.

Finally, we comment that the definition of a PCP of proximity can be extended by providing the verifier with part of the input in an explicit form. That is, referring to Definition 2.7, we let $x = (x', x'')$, and provide V with explicit input $(x', 1^{|x|})$ and input-oracle x'' (rather than with explicit input $1^{|x|}$ and input-oracle x). Clearly, the extended formulation implies PCP as a special case (i.e., $x'' = \lambda$). More interestingly, an extended PCP of proximity for a set of pairs R (e.g., the witness relation of an NP-set), yields a PCP for the set $S \stackrel{\text{def}}{=} \{x' : \exists x'' \text{ s.t. } (x', x'') \in R\}$.

2.3.4 Relating locally testable codes and proofs

Locally testable codes can be thought of as the combinatorial counterparts of the complexity theoretic notion of locally testable proofs (PCPs). This perspective raises the question of whether one of these notions implies (or is useful towards the understanding of) the other.

Do PCPs imply locally testable codes? The use of codes with features related to local testability is implicit in known PCP constructions. Furthermore, the known constructions of locally testable proofs (PCPs) provides a transformation of *standard proofs* (for say SAT) to *locally testable proofs* (i.e., PCP-oracles), such that transformed strings are accepted with probability one by the PCP verifier. Specifically, denoting by S_x the set of standard proofs referring to an assertion x, there exists a polynomial-time mapping f_x of S_x to $R_x \stackrel{\text{def}}{=} \{f_x(y) : y \in S_x\}$ such that for every $\pi \in R_x$ it holds that $\Pr[V^\pi(x) = 1] = 1$, where V is the PCP verifier. Moreover, starting from different standard proofs, one obtains locally testable proofs that are far apart, and hence constitute a good code (i.e., for every x and every $y \neq y' \in S_x$, it holds that $\Delta(f_x(y), f_x(y')) \geq \Omega(|f_x(y)|)$). It is tempting to think that the PCP verifier yields a codeword tester, but this is not really the case. Note that Definition 2.5 requires rejection of strings that are far from any valid proof (i.e., any string far from Π_x), but it is not clear that the only valid proofs (w.r.t V) are those in R_x (i.e., the proofs obtained by the transformation f_x of standard proofs (in S_x) to locally testable ones).[14] In fact, the standard PCP constructions accept also valid proofs that are not in the range of the corresponding transformation (i.e., f_x); that is, Π_x as in Definition 2.5 is a strict subset of R_x (rather than $\Pi_x = R_x$). We comment that most known PCP constructions can be (non-trivially)[15]

[13] That is, given a verifier V as in Definition 2.7, and denoting by n and $p = p(n)$ the sizes of the two parts of its oracle, we consider proofs of length $t \cdot n + p$, where $t = p/o(n)$ (e.g., $t = (p/n) \cdot \log n$). We consider a verifier V' with syntax as in Definition 2.6 that, on input 1^n and oracle access to $w = (u_1, ..., u_t, v) \in \{0,1\}^{t \cdot n + p}$, where $u_i \in \{0,1\}^n$ and $v \in \{0,1\}^p$, selects uniformly $i \in \{1, ..., t\}$ and invokes $V^{u_i, v}(1^n)$. In addition, V' performs a number of repetition tests that is inversely proportional to the proximity parameter, where in each test V' selects uniformly $i, i' \in \{1, ..., t\}$ and $j \in \{1, ..., n\}$ and checks that u_i and $u_{i'}$ agree on their j-th bit. Thus, V' essentially emulates the PCP of proximity V, and the fact that V satisfies Definition 2.7 can be captured by saying that V' satisfies Definition 2.6.

[14] Let alone that Definition 2.4 refers only to the case of false assertions, in which case all strings are far from a valid proof (which does not exist).

[15] The interested reader is referred to [32, Sec. 5.2] for a discussion of typical problems that arise.

modified to yield $\Pi_x = R_x$, and thus to yield a locally testable code (but this is not necessarily the best way to design locally testable codes, see one alternative in §2.3.3).

Do locally testable codes PCPs? Saying that locally testable codes are the combinatorial counterparts of locally testable proofs (PCPs), raises the expectation (or hope) that it would be easier to construct locally testable codes than it is to construct PCPs. The reason being that combinatorial objects (e.g., codes) should be easier to understand than complexity theoretic ones (e.g., PCPs). Indeed, this feeling was among the main motivations of Goldreich and Sudan, and their first result (cf. [32, Sec. 3]) was along this vein: They showed a relatively simple construction (i.e., simple in comparison to PCP constructions) of a locally testable code of T1-nearly linear length. Unfortunately, their stronger result, providing a locally testable code of T2-nearly linear length is obtained by constructing (cf. [32, Sec. 4]) and using (cf. [32, Sec. 5]) a T2-nearly linear locally testable proof (i.e., a PCP). Subsequent works [18, 15] have followed this route, and only the recent work of Ben-Sasson and Sudan [17] (which achieves a more relaxed notion of local testability) reversed the course to the "right one": First codes are constructed, and next they (or actually their analysis) are used towards the construction of proofs (rather than the other way around).

2.3.5 Motivation to the study of short locally testable codes and proofs

Local testability offers an extremely strong notion of efficient testing: The tester makes only a constant number of bit probes, and determining the probed locations (as well as the final decision) is typically done in time that is poly-logarithmic in the length of the probed object.

The length of an error-correcting code is widely recognized as one of the two most fundamental parameters of the code (the second one being its distance). In particular, the length of the code is of major importance in applications, because it determines the overhead involved in encoding information.

As argued in the Introduction, the same considerations apply also to proofs. However, in the case of proofs, this obvious point was blurred by the indirect, unexpected and highly influential applications of PCPs to the theory of approximation algorithms. In our view, the significance of locally testable proofs (or PCPs) extends far beyond their applicability to deriving non-approximability results. The mere fact that proofs can be transformed into a format that supports super-fast probabilistic verification is remarkable. From this perspective, the question of how much redundancy is introduced by such a transformation is a fundamental one. Furthermore, locally testable proofs (i.e., PCPs) have been used not only to derive non-approximability results but also for obtaining positive results (e.g., CS-proofs [36, 39] and their applications [8, 21]), and the length of the PCP affects the complexity of those applications.

In any case, the length of PCPs is also relevant to non-approximability results; specifically, it affects their *tightness with respect to the running time*. For example, suppose (exact) SAT has complexity $2^{\Omega(n)}$. The original PCP Theorem [5, 4] only implies that approximating MaxSAT requires time 2^{n^α}, for some (small) $\alpha > 0$. The work of [40] makes α arbitrarily close to 1, whereas the results of [32, 18] further improve the lower-bound to $2^{n^{1-o(1)}}$. We mention that it is currently not known whether these improved lower-bounds can be achieved simultaneously with optimal approximation ratios, but the hope is that this can eventually be done.

2.4 A confused history

There is a great deal of confusion regarding credits for some of the definitions presented in this section.[16] We refer mainly to the definition of locally testable codes. This definition (or at least a related notion)[17] is arguably implicit in [7] as well as in subsequent works on PCP (see §2.3.4). Furthermore, the definition of locally testable codes has appeared independently in the works of Friedl and Sudan [28] and Rubinfeld and Sudan [41] as well as in the PhD Thesis of Arora [3].

3 Results and Ideas

We review the know constructions of locally testable codes and proofs, starting from codes and proofs of exponential length and concluding with codes and proofs of nearly linear length.

3.1 The mere existence of locally testable codes and proofs

The mere existence of locally testable codes and proofs, regardless of their length, is non-obvious. Thus, we start by recalling the simplest constructions known.

3.1.1 The Hadamard Code is locally testable

The simplest example of a locally testable code (of constant relative distance) is the Hadamard code. This code, denoted $\mathrm{C}_{\mathrm{Had}}$, maps $x \in \{0,1\}^k$ to a string, of length $n = 2^k$, that provides the evaluation of all GF(2)-linear functions at x; that is, the coordinates of the codeword are associated with linear functions $\ell(z) = \sum_{i=1}^{k} \ell_i z_i$ and so $\mathrm{C}_{\mathrm{Had}}(x)_\ell = \ell(x) = \sum_{i=1}^{k} \ell_i x_i$. Testing whether a string $w \in \{0,1\}^{2^k}$ is a codeword reduces to linearity testing. This is the case because w is a codeword of $\mathrm{C}_{\mathrm{Had}}$ if and only if, when viewed as a function $w : \{0,1\}^k \to \{0,1\}$, it is linear (i.e., $w(z) = \sum_{i=1}^{k} c_i z_i$ for some c_i's or equivalently $w(y+z) = w(y) + w(z)$ for all y, z). Specifically, local testability is achieved by uniformly selecting $y, z \in \{0,1\}^k$ and checking whether $w(y+z) = w(y) + w(z)$. The analysis of this natural tester, due to Blum, Luby and Rubinfeld [19], turned out to be highly complex (cf. [19, 6, 25, 12, 13, 10]). In particular, it is known that if w is δ-far from linear then the aforementioned (3-query) test rejects with probability at least $\Gamma(\delta)$, where the function $\Gamma : [0, 0.5] \to [0, 1]$ is defined as follows:

$$\Gamma(x) \stackrel{\mathrm{def}}{=} \begin{cases} 3x - 6x^2 & 0 \le x \le 5/16 \\ 45/128 & 5/16 \le x \le 45/128 \\ x & 45/128 \le x \le 1/2. \end{cases}$$

The above lower bound is composed of three different bounds with "phase transitions" at $x = \frac{5}{16}$ and $x = \frac{45}{128}$. It was shown in [10] that this combined lower bound is close to the best one possible. We believe is that this strange behavior of the rejection probability is a strong indication of the non-triviality of the nature of this "innocent looking" test.

[16]Some confusion exists also with respect to some of the results and constructions described in Section 3, but in comparison to what is going to be discussed here the latter confusion is minor.

[17]The related notion refers to the following relaxed notion of codeword testing: For two fixed good codes $\mathrm{C}_1 \subseteq \mathrm{C}_2 \subset \{0,1\}^n$, one has to accept (with high probability) every codeword of C_1, but reject (with high probability) every string that is far from being a codeword of C_2. Indeed, our definitions refer to the special (natural) case that $\mathrm{C}_2 = \mathrm{C}_1$, but the more general case suffices for the construction of PCPs (and is implicitly achieved in most of them).

Other codes. We mention that Reed-Muller Codes of constant order are also locally testable [1]. These codes have sub-exponential length, but are quite popular in practice. The Long Code is also locally testable [11], but this code has double-exponential length (and was introduced merely for the design of PCPs). Finally, we mention that random linear codes (of linear length) require any codeword tester to read a linear number of bits of the codeword [16], thus providing an additional indication to the non-triviality of local testability.

3.1.2 The Hadamard-Based PCP of ALMSS

The simplest example of a locally testable proof (for a set not known to be in BPP) is the "inner verifier" of the PCP construction of Arora, Lund, Motwani, Sudan and Szegedy [4], which in turn is based on the Hadamard code. Specifically, proofs of the satisfiability of a given system of quadratic equations over GF(2) are presented by providing a Hadamard encoding of the outer-product of a satisfying assignment (i.e., a satisfying assignment $\alpha \in \{0,1\}^n$ is presented by $\mathrm{C}_{\mathrm{Had}}(\beta)$, where $\beta = (\beta_{i,j})_{i,j\in[n]}$ and $\beta_{i,j} = \alpha_i\alpha_j$). Given an alleged proof $\pi \in \{0,1\}^{2^{n^2}}$, the proof-tester proceeds as follows:

1. Tests that π is indeed a codeword of the Hadamard Code. If the test passes then w is close to some $\mathrm{C}_{\mathrm{Had}}(\beta)$, for an arbitrary $\beta = (\beta_{i,j})_{i,j\in[n]}$.

2. Tests that the aforementioned β is indeed an outer-product of some $\alpha \in \{0,1\}^n$. Note that the Hadamard encoding of α is supposed to be part of the Hadamard encoding of β (because $\sum_{i=1}^n c_i\alpha_i = \sum_{i=1}^n c_i\alpha_i^2$ is supposed to equal $\sum_{i=1}^n c_i\beta_{i,i}$). So we would like to test that the latter codeword matches the former one. Specifically, we wish to test whether $(\beta_{i,j})_{i,j\in[n]}$ equals $(\alpha_i\alpha_j)_{i,j\in[n]}$ (i.e., the equality of two matrices). This can be done by uniformly selecting $(r_1,...,r_n),(s_1,...,s_n) \in \{0,1\}^n$, and comparing $\sum_{i,j} r_i s_j \beta_{i,j}$ and $\sum_{i,j} r_i s_j \alpha_i\alpha_j = (\sum_i r_i\alpha_i)(\sum_j s_j\alpha_j)$.

   The above would have been fine if $w = \mathrm{C}_{\mathrm{Had}}(\beta)$, but we only know that w is close to $\mathrm{C}_{\mathrm{Had}}(\beta)$. The Hadamard encoding of α is a tiny part of the latter, and so we should not try to retrieve the latter directly (because this tiny part may be totally corrupted). Instead, we use the paradigm of self-correction (cf. [19]): In general, for any fixed $c = (c_{i,j})_{i,j\in[n]}$, whenever we wish to retrieve $\sum_{i=1}^n c_{i,j}\beta_{i,j}$, we uniformly select $r = (r_{i,j})_{i,j\in[n]}$ and retrieve both $w(r)$ and $w(r+c)$. Thus, we obtain a self-corrected value of $w(c)$; that is, if w is δ-close to $\mathrm{C}_{\mathrm{Had}}(\beta)$ then $w(r+c) - w(r) = \sum_{i=1}^n c_{i,j}\beta_{i,j}$ with probability at least $1-2\delta$.

   Using self-correction, we indirectly obtain bits in $\mathrm{C}_{\mathrm{Had}}(\alpha)$, for $\alpha = (\alpha_i)_{i\in[n]} = (\beta_{i,i})_{i\in[n]}$. Similarly, we can obtain any other desired bit in $\mathrm{C}_{\mathrm{Had}}(\beta)$, which in turn allows us to test whether $(\beta_{i,j})_{i,j\in[n]} = (\alpha_i\alpha_j)_{i,j\in[n]}$. In fact, we are checking whether $(\beta_{i,j})_{i,j\in[n]} = (\beta_{i,i}\beta_{j,j})_{i,j\in[n]}$, by comparing $\sum_{i,j} r_i s_j \beta_{i,j}$ and $(\sum_i r_i\beta_{i,i})(\sum_j s_j\beta_{j,j})$, for randomly selected $(r_1,...,r_n),(s_1,...,s_n) \in \{0,1\}^n$.

3. Finally, we need to check whether the aforementioned α satisfies the given system of equations. Towards this end, we uniformly selects a linear combination of the equations, and check whether α satisfies the resulting (single) equation. Note that the value of the corresponding linear expression (in quadratic (and linear) forms) appears as a bit of the Hadamard encoding of β, but again we retrieve it from w by using self correction.

One key observation underlying the analysis of Steps 2 and 3 is that for $(u_1,...,u_n) \neq (v_1,....,v_n) \in \{0,1\}^n$, if we uniformly select $(r_1,....,r_n) \in \{0,1\}^n$ then $\Pr[\sum_i r_i u_i = \sum_i r_i v_i] = 1/2$. Similarly, for n-by-n matrices $A \neq B$, when $r, s \in \{0,1\}^n$ are uniformly selected (vectors), it holds that $\Pr[As = Bs] = 2^{-\mathrm{rank}(A-B)}$ and it follows that $\Pr[rAs = rBs] \leq 3/4$.

3.2 Locally testable codes and proofs of polynomial length

The constructions presented in Section 3.1 have exponential length in terms of the relevant parameter (i.e., the amount of information being encoded in the code or the length of the assertion being proved). Achieving local testability by codes and proofs that have polynomial length turns out to be even more challenging.

3.2.1 Locally testable codes of quadratic length

A direct interpretation of *low-degree tests* (cf. [6, 7, 29, 41, 28]), proposed by Friedl and Sudan [28] and Rubinfeld and Sudan [41], yields a locally testable code of quadratic length over a *sufficiently large alphabet*. Similar (and actually better) results for *binary* codes required additional ideas, and have appeared only later (cf. [32]). We sketch both constructions below, starting with locally testable codes over very large alphabets (which are defined analogously to the binary case).

We will consider a code $\mathrm{C} : \Sigma^k \to \Sigma^n$ of linear distance, with $|\Sigma| \gg k$ and $n > k^2$. For parameters $m \ll d < \log k$ (such that $k < d^m$), consider a finite field F of size $O(d)$ and an alphabet $\Sigma = F^{d+1}$. Viewing the information as a m-variant polynomial p of total degree d over F, we encode it by providing its value on all possible lines over F^m, where each such line is defined by two points in F^m. Actually, the value of p on such a line can be represented by a univariant polynomial of degree d. Thus, the code maps $\log_2 |F|^{\binom{m+d}{d}} > (d/m)^m \log |F|$ bits of information (which may be viewed as $k \stackrel{\text{def}}{=} (d/m)^m/(d+1) \approx d^{m-1}/m^m$ long sequences over $\Sigma = F^{d+1}$) to sequences of length $n \stackrel{\text{def}}{=} |F|^{2m} = O(d)^{2m}$ over Σ. Note that the smaller m, the better the rate (i.e., relation of n to k) is, but this comes at the expense of using a larger alphabet. In particular, we consider two instantiations:

1. Using $d = m^m$, we get $k \approx m^{m^2-2m}$ and $n = m^{2m^2+o(m)}$, which yields $n \approx \exp(\sqrt{\log k}) \cdot k^2$ and $\log |\Sigma| = \log |F|^{d+1} \approx d \log d \approx \exp(\sqrt{\log k})$.

2. Letting $d = m^c$ for any constant $c > 1$, we get $k \approx m^{(c-1)m}$ and $n = m^{2cm+o(m)}$, which yields $n \approx k^{2c/(c-1)}$ and $\log |\Sigma| \approx d \log d \approx (\log k)^c$.

As for the codeword tester, it uniformly selects two intersecting lines and checks that the corresponding univarinat polynomials agree on the point of intersection. Thus, this tester makes two queries (to an oracle over the alphabet Σ). The analysis of this tester reduces to the analysis of the corresponding low degree test, undertaken in [4, 40].

The above tester uses only two queries, but the entire description (which refers to codes over a large alphabet) deviates from the bulk of our treatment, which has focused on a binary alphabet. We comment that 2-query locally testable *binary* codes are essentially impossible (cf., [14]), but we have already seem that 3-query tests are possible. A natural way of reducing the alphabet size of codes is via the well-known paradigm of *concatenated codes* [26].[18] However, local testability can be maintained only in special cases. In particular, observe that, for each of the two queries

[18] A concatenated code is obtained by encoding the symbols of an "outer code" (using the coding method of the "inner code"). Specifically, let $\mathrm{C}_1 : \Sigma_1^{k_1} \to \Sigma_1^{n_1}$ be the outer code and $\mathrm{C}_2 : \Sigma_2^{k_2} \to \Sigma_2^{n_2}$ be the inner code, where $\Sigma_1 \equiv \Sigma_2^{k_2}$. Then, the concatenated code $\mathrm{C} : \Sigma_2^{k_1 k_2} \to \Sigma_2^{n_1 n_2}$ is obtained by $\mathrm{C}(x_1, ..., x_{k_1}) = (\mathrm{C}_2(y_1), ..., \mathrm{C}_2(y_{n_1}))$, where $x_i \in \Sigma_2^{k_2} \equiv \Sigma_1$ and $(y_1, ..., y_{n_1}) = \mathrm{C}_1(x_1, ..., x_{k_1})$. Using a good inner code for relatively short sequences, allows to transform good codes for a large alphabet into good codes for a smaller alphabet.

made by the tester of C, the tester does not need the entire polynomial represented in $\Sigma = F^{d+1}$, but rather only its value at a specific point. Thus, encoding Σ by an error correcting code that supports recovery of the said value while using a constant number of probes will do.[19] In particular, Goldreich and Sudan used an encoding of $F^{d+1} = F^{h^e}$ by sequences of length $|F|^{eh}$ over F, and provided a testing and recovery procedure that makes $O(e)$ queries [32, Sec. 3.3]. We mention that the case of $e = 1$ and $|F| = 2$ corresponds to the Hadamard code, and that bigger constant e allow for shorter codes. The resulting concatenated code, C′, is a locally testable code over F, and has length $n \cdot O(d)^{eh} = n \cdot \exp((e \log d) \cdot d^{1/e})$. Using constant $e = 2c$ and setting $d = m^c \approx (\log k)^c$, we get $n \approx k^{2c/(c-1)} \cdot \exp(\widetilde{O}(\log k)^{1/2})$ and $|F| = \text{poly}(\log k)$. Finally, a *binary* locally testable code is obtained by concatenating C′ with the Hadamard code, while noting that the latter supports a "local recovery" property that suffices to emulate the tester for C′. In particular, the tester of C′ merely checks a linear (over F) equation referring to a constant number of F-elements, and for $F = GF(2^\ell)$, this can be emulated by checking *related* random linear combinations of the bits representing these elements, which in turn can be local recovered (or rather self-corrected) from the Hadamard code. The final result is a locally testable (binary) code of nearly quadratic length.[20]

3.2.2 Locally testable proofs of polynomial length: The PCP Theorem

The case of proofs is far more complex: Achieving locally testable proof of polynomial length is essentially the contents of the celebrated PCP Theorem of Arora, Lund, Motwani, Safra, Sudan and Szegedy [5, 4]. The construction is analogous to (but far more complex than) the one presented in the case of codes:[21] First one constructs proofs over a large alphabet, and next one composes such proofs with corresponding "inner" proofs (over a smaller alphabet, and finally a binary one).

The first step is to introduce the following NP-complete problem. The input to the problem consists of a finite field F, a subset $H \subset F$ of size $\lfloor |F|^{1/15} \rfloor$, an integer $m < |H|$, and a $(3m+4)$-variant polynomial $P : F^{3m+4} \to F$ of total degree $3m|H| + O(1)$. The problem is to determine whether there exists an m-variant ("assignment") polynomial $A : F^m \to F$ of total degree $m|H|$ such that $P(x, z, y, \tau, A(x), A(y), A(z)) = 0$ for every $x, y, z \in H^m$ and $\tau \in \{0,1\}^3 \subset H$. Note that the problem-instance can be explicitly described by a sequence of $|F|^{3m+4} \log_2 |F|$ bits, whereas the solution sought can be explicitly described by a sequence of $|F|^m \log_2 |F|$ bits. We comment that the NP-completeness of the aforementioned problem can be proved by a reduction from 3SAT, by identifying the variables of the formula with H^m and essentially letting P be a low-degree extension of a function $f : H^{3m} \times \{0,1\}^3 \to \{0,1\}$ that encodes the structure of the formula (by considering all possible 3-clauses). In fact, the resulting P has degree $|H|$ in each of the first $3m$ variables and constant degree in each of the other variables, and this fact can be used to improve the parameters below (but not in a fundamental way).

The proof that P satisfies the aforementioned condition consists of an m-variant polynomial $A : F^m \to F$ (which is supposed to be of total degree $m|H|$) as well as $3m + 4$ auxiliary polynomials $A_i : F^{3m+1} \to F$, for $i = 1, ..., 3m + 1$ (each supposedly of degree $(3m|H| + O(1)) \cdot m|H|$). The polynomial A is supposed to satisfy the conditions of the problem, and in particular $P(x, z, y, \tau, A(x), A(y), A(z)) = 0$ should hold for every $x, y, z \in H^m$ and $\tau \in \{0,1\}^3 \subset H$. Fur-

[19] Indeed, this property is related to locally decodable codes, to be discussed in Section 4. Here we need to recover one out of $|F|$ specific linear combinations of the encoded $(d+1)$-long sequence of F-symbols. In contrast, locally decodable refers to recovering one out of the original F-symbols of the $(d+1)$-long sequence.

[20] Actually, the aforementioned result is only implicit in [32], because Goldreich and Sudan apply these ideas directly to a truncated version of the low-degree based code.

[21] Our presentation reverses the historical order in which the corresponding results (for codes and proofs) were achieved. That is, the constructions of locally testable proof of polynomial length predated the coding counterparts.

thermore, $A_0(x,z,z,\tau) \stackrel{\rm def}{=} P(x,z,y,\tau,A(x),A(y),A(z))$ should vanish on H^{3m+1}. The auxiliary polynomials are given to assist the verification of the latter condition. In particular, it should be the case that A_i vanishes on F^iH^{3m+1-i}, a condition that is easy to test for A_{3m+1} (assuming that A_{3m+1} is a low degree polynomial). Checking that A_{i-1} agrees with A_i on $F^{i-1}H^{3m+1-(i-1)}$, for $i=1,...,3m+1$, and that all A_i's are low degree polynomials, establishes the claim for A_0. Thus, testing an alleged proof $(A, A_1, ..., A_{3m+1})$ is performed as follows:

1. Testing that A is a polynomial of total degree $m|H|$. This is done by selecting a random line through F^m, and testing whether A restricted to this line agrees with a degree $m|H|$ univariant polynomial.

2. Testing that, for $i=1,...,3m+1$, the polynomial A_i is of total degree $d \stackrel{\rm def}{=} (3m|H|+O(1)) \cdot m|H|$. Here we select a random line through F^{3m+1}, and test whether A_i restricted to this line agrees with a degree d univariant polynomial.

3. Testing that, for $i=1,...,3m+1$, the polynomial A_i agrees with A_{i-1} on $F^{i-1}H^{3m+1-(i-1)}$. This is done by uniformly selecting $r'=(r_1,...,r_{i-1}) \in F^{i-1}$ and $r''=(r_{i+1},...,r_{3m+1}) \in F^{3m+1-i}$, and comparing $A_{i-1}(r',e,r'')$ to $A_i(r',e,r'')$, for every $e \in H$. In addition, we check that both functions when restricted to the axis-parallel line $(r',\cdot,r'')$ agree with a univariant polynomial of degree d.[22] We stress that the values of A_0 are computed according to the given polynomial P by accessing A at the adequate locations (i.e., by definition $A_0(x,z,z,\tau) = P(x,z,y,\tau,A(x),A(y),A(z))$).

4. Testing that A_{3m+1} vanishes on F^{3m+1}. This is done by uniformly selecting $r \in F^{3m+1}$, and testing whether $F(r)=0$.

The above description (which follows [42, Apdx. C]) is somewhat different than the original presentation in [4], which in turn follows [6, 7, 25].[23] The above tester may be viewed as making $O(m|F|)$ queries to an oracle over the alphabet F, or alternatively as making $O(m|F|\log|F|)$ binary queries.[24] Note that we have already obtained a highly non-trivial tester. It makes $O(m|F|\log|F|)$ queries in order to verify a claim regarding an input of length $n \stackrel{\rm def}{=} |F|^{3m+4}\log_2|F|$. Using $m = \log n/\log\log n$, $|H| = \log n$ and $|F| = \mathrm{poly}(\log n)$, we have obtained a tester of poly-logarithmic query complexity.

To further reduce the query complexity, one invokes the "proof composition" paradigm, introduced by Arora and Safra [5]. Specifically, one composes an "outer" tester (as described above) with an "inner" tester that checks the residual condition that the "outer" tester determines for the answers it obtains. This composition is more problematic than one suspects, because we wish the "inner" tester to perform its task without reading its entire input (i.e., the answers to the "outer" tester). This seems quite paradoxical, as how can the "inner" tester operate without reading its entire input. The problem can be resolved by using a "proximity tester" (i.e., a PCP of proximity) as an "inner" tester, provided that it suffices to have such a proximity test (for the answers to the "outer" tester).

- One approach, introduced in [4], is to convert the "outer" tester into one that makes a constant number of queries over some larger alphabet, and furthermore have the answer be presented

[22] Thus, effectively, we are self-correcting the values at H (on the said line), based on the values at F (on that line).

[23] The point is that the sum-check, which originates in [38], is replaced by an analogous process (which happens to be non-adaptive).

[24] Another alternative perspective is obtained by applying so-called parallelization (cf. [37, 4]). The result is a test making a constant number of queries that are each answered by strings of length $\mathrm{poly}(|F|)$.

in an error correcting format. The implementation of this approach consists of two steps and is based on some specifics. The first step is to convert the "outer" tester into one that makes a constant number of queries over some larger alphabet. This step uses the so-called parallelization technique (cf. [37, 4]). Next, one applies an error correcting code to these $O(1)$ longer answers, and assumes that the "proximity tester" can handle inputs presented in this format (i.e., that it can test an input that is presented by an encoding of a constant number of its parts).[25]

- An alternative approach, pursued and advocated in [15], is to take advantage of the specific structure of the queries, "bundle" the answers together and furthermore show that the "bundled" answers are "robust" in a sense that fits proximity testing. In particular, the (generic) parallelization step is avoided, and is replaced by a closer analysis of the specific (outer) tester.

We will demonstrate the latter approach next.

First, we show how the queries of the aforementioned tester can be "bundled" (into a constant number of bundles). In particular, we consider the following "bundling" that accommodates all types of tests (and in particular the $m+1$ different sub-tests performed in Steps 2 and 3). Consider

$$B(x_1, ..., x_{3m+1}) = (A_1(x_1, x_2, ..., x_{3m+1}), A_2(x_2, ..., x_{3m+1}, x_1), ..., A_{3m+1}(x_{3m+1}, x_1, ..., x_{3m}))$$

and perform all $3m+1$ tests of Step (3) by selecting uniformly $(r_2, ..., r_{3m+1}) \in F^{3m}$ and querying B at $(e, r_2, ..., r_{3m+1})$ and $(r_{3m+1}, e, ..., r_{3m})$ for all $e \in F$. Thus, all $3m+1$ tests of Step (3) can be performed by retrieving the values of B on a single *axis parallel* random line through F^{3m+1}. Furthermore, note that all $3m+1$ tests of Step (2) can be performed by retrieving the values of B on a single (arbitrary) random line through F^{3m+1}. Finally, observe that the tests are "robust" in the sense that if, for some i, the function A_i is (say) 0.01-far from satisfying the condition (i.e., being low-degree or agreeing with A_{i-1}) then with constant probability many of the values of A_i on an adequate random line will not fit to what is needed. This robustness property is inherited by B, as well as by B' (resp., A') that is obtained by applying a good binary error-correcting code on B (resp., on A). Thus, we may replace A and the A_i's by A' and B', and conduct all all tests by making $O(m^2|F|\log|F|)$ queries to $A' : F^m \times [O(\log|F|)] \to \{0,1\}$ and $B' : F^{3m+1} \times [O(\log|F|^{3m+1})] \to \{0,1\}$. The *robustness property* asserts that if the original polynomial P had no solution (i.e., an A as above) then the answers obtained by the tester will be far from satisfying the residual decision predicate of the tester.

Once the robustness property of the resulting ("outer") tester fits the proximity testing feature of the "inner tester", composition is possible. Indeed, we compose the "outer" tester with an "inner tester" that checks whether the residual decision predicate of the "outer tester" is satisfies. The benefit of this composition is that the query complexity is reduced from poly-logarithmic to polynomial in a double-logarithm. At this point we can afford the Hadamard-Based proof tester (because the overhead in the proof complexity will only be exponential in a polynomial in a double-logarithmic function), and obtain a locally testable proof of polynomial length.

[25] The aforementioned assumption holds trivially in case one uses a generic "proximity tester" (i.e., a PCP of proximity or an Assignment Tester) as done in [23]. But the aforementioned approach can be (and was in fact originally) applied with a specific "proximity tester" that can only handle inputs presented in one specific format (cf. [4]).

3.3 Locally testable codes and proofs of nearly linear length

We now move on to even *shorter* codes and proofs; specifically, codes and proofs of *nearly linear length*. The latter term has been given quite different interpretations, and we start by sorting these out.

3.3.1 Types of nearly linear functions

A few common interpretations of this term are listed below (going from the most liberal to the most strict one).

T1-nearly linear: A very liberal notion, at the verge of an abuse of the term, refers to a sequence of functions $f_\epsilon : \mathsf{N} \to \mathsf{N}$ such that, for every $\epsilon > 0$, it holds that $f_\epsilon(n) \leq n^{1+\epsilon}$. That is, each function is actually of the form $n \mapsto n^c$, for some constant $c > 1$, but the sequence as a whole can be viewed as approaching linearity.

The PCP of Polishchuk and Spielman [40] and the simpler locally testable code of Goldreich and Sudan [32, Thm. 2.4] have nearly linear length in this sense.

T2-nearly linear: A more reasonable notion of nearly linear functions refers to individual functions f such that $f(n) = n^{1+o(1)}$. Specifically, for some function $\epsilon : \mathsf{N} \to [0,1]$ that goes to zero, it holds that $f(n) \leq n^{1+\epsilon(n)}$. Common sub-types include the following:

1. $\epsilon(n) = 1/\log\log n$.
2. $\epsilon(n) = 1/(\log n)^c$ for some $c \in (0,1)$.

   The currently best locally testable codes and proofs [32, 18, 15] have nearly linear length in this sense.
3. $\epsilon(n) = \exp((\log\log\log n)^c)/\log n$ for some $c \in (0,1)$.

Indeed, the case in which $\epsilon(n) = O(\log\log n)/\log n$ (or so) deserves a special category.

T3-nearly linear: The strongest notion interprets near-linearity as linearity up to a poly-logarithmic (or quasi-poly-logarithmic) factor. In the former case $f(n) \leq \mathrm{poly}(\log n) \cdot n$, which corresponds to the case of $f(n) \leq n^{1+\epsilon(n)}$ with $\epsilon(n) = O(\log\log n)/\log n$, whereas the latter case corresponds to $\epsilon(n) = \mathrm{poly}(\log\log n)/\log n$ (i.e., in which case $f(n) \leq (\log n)^{\mathrm{poly}(\log\log n)} \cdot n$).

Using the above notation, we summarize the state of the art with respect to local testability of codes and proofs.

3.3.2 Local testability with nearly linear length

Currently, locally testable codes and proofs of nearly linear length are known when nearly linear is interpreted as Type T2 (i.e., T2-nearly linear). More generally, we have:

Theorem 3.1 (Ben-Sasson, Goldreich, Harsha, Sudan and Vadhan [15]): *There exists a universal constant $c > 2$ such that for every function $q : \mathsf{N} \to \mathsf{N}$ satisfying $2c \leq q(k) \leq \frac{2c\log\log k}{\log\log\log k}$ there exists a q-locally testable proof of length $F_q(k) \cdot k$ for satisfiability (of formulae of length k), where*

$$F_q(k) \stackrel{\text{def}}{=} \exp\left[\left(\frac{q(k)}{c} + (\log k)^{\frac{c}{q(k)}}\right) \cdot (\log\log k) + (\log k)^{\frac{2c}{q(k)}} + \frac{q(k)^2}{c^2} \cdot \mathrm{poly}\log\log\log k\right] \qquad (1)$$

The same length bound holds for q-locally testable codes, where k denotes the length of the information being encoded.

Let us derive two extreme cases of Theorem 3.1, while setting $t = q(k)/c$.

1. *Constant query complexity*: For $t \in [2, ..., \frac{0.99 \log\log k}{\log\log\log k}]$, we have $(\log k)^{\frac{1}{t}} > (\log\log k)^{1/0.99}$ and so $F_t(k) = \exp((\log k)^{\frac{2}{t}})$. In particular, for any constant t, we get *locally testable proofs and codes* (i.e., $c \cdot t$-locally testable proofs and codes) *of length* $\exp((\log k)^{\frac{2}{t}}) \cdot k = k^{1+\epsilon(k)}$, *where* $\epsilon(k) = 1/(\log k)^{1-\frac{2}{t}}$.

2. *T3-nearly linear length*: For $t \geq \frac{1.01 \log\log k}{\log\log\log k}$, we have $(\log k)^{\frac{1}{t}} \leq (\log\log k)^{1/1.01}$ and so $F_t(k) = \exp(t^2 \cdot \text{poly}\log\log\log k) = \exp(\widetilde{O}(\log\log k)^2)$. In particular, setting $t = \frac{2\log\log k}{\log\log\log k}$, we get $o(\log\log k)$*-locally testable proofs and codes of length* $\exp(\widetilde{O}(\log\log k)^2) \cdot k$.

For an even stricter notion of T3-nearly linear (i.e., a poly-logarithmic factor rather than a quasi-poly-logarithmic one), testers of poly-logarithmic query complexity are known.

query complexity	length overhead	comments
$\text{poly}(\log k)$	$\text{poly}(\log k)$	Theorem 3.2.
$o(\log\log k)$	$\exp(\text{poly}(\log\log k))$	These are two extreme cases
Any constant q	$\exp((\log k)^{O(1/q)})$	of Theorem 3.1.

Figure 1: The best known $q(\cdot)$-locally testable codes and proofs

Theorem 3.2 (Ben-Sasson and Sudan [17]): *There exists a poly-logarithmic function* $f : \mathsf{N} \to \mathsf{N}$ *such that there exist* f*-locally testable codes and proofs of length* $f(k) \cdot k$, *where* k *denotes the length of the actual information* (i.e., the assertion in case of proofs and the encoded information in case of codes).

The known results are summarized in Figure 1, where k is as in Theorems 3.1 and 3.2. The ultimate goal may be to obtain locally testable (i.e., $O(1)$-locally testable) codes and proofs of T3-nearly linear length. Indeed, we conjecture that this is possible.

Conjecture 3.3 *There exist locally testable codes and proofs of length* $\text{poly}(\log k) \cdot k$.

We conjecture that locally testable codes and proofs of (strictly) linear length cannot be achieved.

Conjecture 3.4 *There exist no locally testable codes and proofs of linear length.*

3.3.3 The ideas underlying the constructions

We briefly mention some of the ideas that underly the proofs of Theorems 3.1 and 3.2.

A nearly linear arithmetic representation of SAT. The proof of the PCP Theorem starts by a reduction of 3SAT to an arithmetic problem, but the reduction (as sketched in §3.2.2) represents an n-variable input formula as a binary string of length $O(n^3)$. Thus, this very first step already cubes the length of the constructed proof. An alternative arithmetization, which only incurs a poly-logarithmic increase in length, is obtained by first "embedding" the formula in a de-Bruijn graph such that the variables and clauses are placed at vertices of the opposite ends of the graph (cf. [7, 40]). The specific formula will be encoded in an adequate routing of the variables to the clauses in which they appear, and the arithmetization will "hard-wire" this routing in an adequate polynomial (of the type used in §3.2.2). Extra complications arises when one seeks to perform this process "optimally" (i.e., with the minimal number of variables), which is important when using large fields (as seems required for deriving the results of Theorem 3.1). These difficulties are resolved in different ways in [40] and in [15], respectively.

Derandomizing low-degree tests. Another source of polynomial blow-up in the proof length is the low-degree tests, which play a key role in all PCP constructions (cf. §3.2.2). Recall that to test that a function of the form $f : F^m \to F$ is low degree, we fetched its values on points of a random line. Since a sub-proof will be (eventually) appended per each such a choice (of a line), we will need $|F^{2m}|$ such sub-proofs squaring the size of the original function f. Thus, a derandomization of this test (as done non-constructively in [32] and constructively in [18, 15]) is of key importance. In particular, it turns out that it suffices to consider a set of $\widetilde{O}(|F^{m-1}|)$ lines; specifically, each line is specified by a canonical point (residing on this line) and a slope that belongs to a subset of poly-logarithmic many slopes (out of all $|F^m|$ possible slopes) [18].

Avoiding parallelization. As explained in §3.2.2, parallelization play a key role in all previous PCP constructions, and applying it increases the size of the proof by a factor that is at least proportional to the query complexity of the original PCP. But this is too much in the context of proving Theorem 3.1, and so the alternative "bundling" technique was introduced and used (in [15]) in order to support a new proof composition method (sketched already in §3.2.2). Similarly, other types of packing various polynomials into a single polynomial (by using an auxiliary variable), which were used in prior constructions, have to be avoided.

Unbounded number of proof compositions. As mentioned above, proof composition plays a central role in the construction of PCPs. The reason being that a PCP must satisfy two conflicting conditions; specifically, have relatively small query complexity and still be short. Trying to optimize both complexity measures simultaneously turns out to be very hard, and proof composition allows to make progress based on "non-optimal" constructions. Typically, the more we can apply proof composition, the better. Indeed, significant progress was achieved by using a non-constant (e.g., double-logarithmic) number of proof compositions [15, 23]. In the context of providing short PCPs, the new composition method of [15] has played an important role. The result, stated in Theorem 3.1, is a PCP with query complexity that is linear in the number of proof compositions (denoted t), and length overhead that decreases double-exponentially with this number (i.e., the overhead is essentially $\exp((\log k)^{2/t})$).

Recursive construction of a special purpose PCPP. The aforementioned proof composition paradigm seems to incur an unavoidable poly-logarithmic blow-up in the proof length, per each application. This is the source of the $(\log k)^t$ factor in Eq. (1), where $t = q(k)/c$ is the number of proof compositions. This overhead is due to the fact that we *reduce a specific problem* (i.e., evaluating the residual tester decision regarding the oracle's answers) *to a generic one*, and then arithmetize the latter.[26] An alternative approach was taken in [17], resulting in Theorem 3.2: They first construct a $q(k)$-local codeword tester, for $q(k) = \sqrt{k}$, and then reduce the residual test

[26]Our feeling is that a poly-logarithmic blow-up is unavoidable when reducing a specific problem to a generic one.

to testing the *same codeword property* on sequences of length $q(k)$. Unfortunately, the reduction uses a constant number of recursive calls, and so the end result uses a number of queries that is exponential (rather than linear) in the number of compositions, which in turn is double-logarithmic.

4 Locally Decodable Codes

Locally *decodable* codes are complimentary to local *testable* codes. Recall that the latter are required to allow for super-fast rejection of strings that are far from being codewords (while accepting all codewords). In contrast, in case of locally decodable codes, we are guaranteed that the input is close to a codeword, and are required to recover individual bits of the encoded information based on a *small number of probes* (per recovered bit). As in case of local testability, the case when the operation (in this case decoding) is performed based on a *constant number of probes* is of special interest.

Local decodability is of natural practical appeal, which in turn provides additional motivation for local testability. The point being that it makes little sense to try recover part of the data, in case the codeword is too corrupted. Thus, one should first apply local testability to check that the received codeword is not too corrupted, and apply local decodability only in case the codeword test passes.

4.1 Definitions

We follow the conventions of Section 2.1, but extend the treatment to codes over any finite alphabet Σ (rather than insisting on $\Sigma = \{0,1\}$). In the following, we use the notation $[k] \stackrel{\text{def}}{=} \{1, 2, ..., k\}$.

Definition 4.1 *Let* $\mathrm{C} : \Sigma^k \to \Sigma^n$ *be a code, and let* $q \in \mathsf{N}$ *and* $\delta \in (0,1)$*. A* q-local δ-decoder for C *is a probabilistic* (non-adaptive) *oracle machine* M *that makes at most* q *queries and satisfies the following condition:*

Local recovery from somewhat corrupted codewords: *For every* $i \in [k]$ *and* $x = (x_1, ..., x_k) \in \Sigma^k$*, and any* $w \in \Sigma^n$ *that is* δ*-close to* $\mathrm{C}(x)$*, on input* i *and oracle access to* w*, machine* M *outputs* x_i *with probability at least* $2/3$*. That is,* $\Pr[M^w(1^k, i) = x_i] > 2/3$*, for any* $w \in \Sigma^n$ *that is* δ*-far from* $\mathrm{C}(x)$*.*

We call q *the* query complexity *of* M*, and* δ *the* proximity parameter*.*

Note that the proximity parameter must be smaller than the covering radius of the code (as otherwise the definition cannot possibly be satisfies (at least for some w and i)). One may strengthen Definition 4.1 by requiring that the bits of an uncorrupted codeword be always recovered correctly (rather than with high probability); that is, for every $i \in [k]$ and $x = (x_1, ..., x_k) \in \Sigma^k$, it must hold that $\Pr[M^{\mathrm{C}(x)}(1^k, i) = x_i] = 1$. Turning to families of codes, we make the following definition (which potentially allows the alphabet to grow with k).

Definition 4.2 *For functions* $n, \sigma : \mathsf{N} \to \mathsf{N}$*, let* $\mathrm{C} = \{\mathrm{C}_k : [\sigma(k)]^k \to [\sigma(k)]^{n(k)}\}_{k \in K}$*. We say that* C *is a* local decodable code *if there exist constants* $\delta > 0$ *and* q *and a machine* M *that is a* q*-local* δ*-decoder for* C_k*, for every* $k \in K$*.*

We mention that locally decodable codes are related to (information theoretic secure) Private Information Retrieval schemes, introduced in [22]. In the latter a user wishes to recover a bit of data from a k-bit long database, copies of which are held by s servers, without revealing any information to any single server. To that end, the user (secretly) communicates with each of the servers, and the issue is to minimize the total amount of communication. As we shall see, certain s-server PIR schemes yield $2s$-locally decodable codes of length exponential in the communication complexity of the PIR.

Related notions of local recovery. The notion of local decodability is a special case of a general notion of local recovery, where one may be required to recover an arbitrary function of the original information based on a constant number of probes to the (corrupted) codeword. The function $f : \Sigma^k \to \{0,1\}^*$ be better restricted in two ways: First it should have a small range (e.g., its range may be Σ), and secondly it should come from a small predetermined set $\mathcal{F}$ of functions. Definition 4.1 may be recast in these terms, by considering the set of projection functions (i.e., $\{f_i : \Sigma^k \to \Sigma\}$ where $f_i(x_1, ..., x_k) = x_i$). We believe that this is the most natural special case of the general notion of local recovery. In §3.2.1 we referred to another special case, where the alphabet is associated with a finite field F and the recovery function $f_e : F^k \to F$ is one out of $|F|$ possible linear functions (specifically, $f_e(x_1, ..., x_k) = \sum_{i=1}^{k} e^{i-1} x_i$, for $e \in F$).[27] Another natural case (also used in §3.2.1) is that of the recovery of (correct) symbols of the codeword, which may be viewed as self-correction. (In this case the set of functions correspond to the functions determining each codeword symbol as a function of the encoded message.)

4.2 Results

The best known locally decodable codes are of sub-exponential length. Specifically, k information bits can be encoded by codewords of length $n = \exp(k^{O(\log\log q)/q \log q})$ that are locally decodable using q bit-probes (cf. [9]). It is conjectured that, for every q there exists an $\epsilon > 0$, such that locally decodability based on q queries (i.e., probes) requires codewords of length $n > \exp(k^{\epsilon})$.

4.2.1 Locally decodable codes of sub-exponential length

For any $d \geq 1$, there is a simple construction of a 2^d-locally 2^{-d-2}-decodable binary code of length $n = 2^{d \cdot k^{1/d}}$. For $h = k^{1/d}$, we identify $[k]$ with $[h]^d$, and view $x \in \{0,1\}^k$ as $(x_{i_1,...,i_d})_{i_1,...,i_d \in [h]}$. We encode x by providing the parity of all $x_{i_1,...,i_d}$ residing in each of the $(2^h)^d$ sub-cubes of $[h]^d$; that is, for every $(S_1, ..., S_d) \in 2^{[h]} \times \cdots \times 2^{[h]}$, we provide $\mathrm{C}(x)_{S_1,...,S_d} = \oplus_{i_1 \in S_1,...,i_d \in S_d} x_{i_1,...,i_d}$. Indeed, the Hadamard code is the special case in which $d = 1$. To recover the value of $x_{i_1,...,i_d}$, at any desired $(i_1, ..., i_d) \in [h]^d$, the decoder uniformly selects $(R_1, ..., R_d) \in 2^{[h]} \times \cdots \times 2^{[h]}$, and recovers the (possibly corrupted) values $\mathrm{C}(x)_{S_1,...,S_d}$, where each S_j either equals R_j or equals $R_j \triangle \{i_j\}$. The key observation is that each of the decoder's queries is uniformly distributed. Thus, with probability at least 3/4, XORing the 2^d answers, yields the desired result (because $\oplus_{S_1 \in \{R_1, R_1 \triangle \{i_1\}\},...,S_d \in \{R_d, R_d \triangle \{i_d\}\}} \mathrm{C}(x)_{S_1,...,S_d}$ equals $\mathrm{C}(x)_{\{i_1\},...,\{i_d\}} = x_{i_1,...,i_d}$).

We comment that a related code (of length $n = 2^{d^d \cdot k^{1/d}}$) allows for recovery based on $d+1$ (rather 2^d) queries. The original presentation, due to [2] (building on [22]), is in terms of PIR schemes (with $s = (d+1)/2$ servers and overall communication $d^d \cdot k^{1/d} = \exp(\widetilde{O}(s)) \cdot k^{1/(2s-1)}$). In particular, in the case $d = 2$, we use two servers, sending (R_1, R_2, R_3) to one and $(R_1 \triangle \{i_1\}, R_2 \triangle \{i_2\}, R_3 \triangle \{i_3\})$ to the other. Upon receiving (S_1, S_2, S_3), each server replies with the bit $\mathrm{C}(x)_{S_1,S_2,S_3} = \oplus_{j_1 \in S_1, j_2 \in S_2, j_3 \in S_3} x_{j_1,j_2,j_3}$, as well as the sequences $(\mathrm{C}(x)_{S_1 \triangle \{1\},S_2,S_3}, ..., \mathrm{C}(x)_{S_1 \triangle \{k^{1/3}\},S_2,S_3})$,

[27] Indeed, the value $f_e(x_1, ..., x_k)$ is the evaluation at e of the polynomial $p(\zeta) = \sum_{i=1}^{k} x_i \zeta^{i-1}$ represented by the coefficients $(x_1, ..., x_k)$.

$(\mathrm{C}(x)_{S_1,S_2\triangle\{1\},S_3},...,\mathrm{C}(x)_{S_1,S_2\triangle\{k^{1/3}\},S_3})$, and $(\mathrm{C}(x)_{S_1,S_2,S_3\triangle\{1\}},...,\mathrm{C}(x)_{S_1,S_2,S_3\triangle\{k^{1/3}\}})$, which allow the user to recover $\mathrm{C}(x)_{S_1\triangle\{i_1\},S_2,S_3}$, $\mathrm{C}(x)_{S_1,S_2\triangle\{i_2\},S_3}$, and $\mathrm{C}(x)_{S_1,S_2,S_3\triangle\{i_1\}}$.

The corresponding locally decodable code is obtained by a generic transformation that applies to any PIR scheme with s servers, in which the user makes uniformly distributed queries of length $\mathsf{qst}(k)$, gets answers of length $\mathsf{ans}(k)$, and recovers the desired value by XORing some predetermined bits contained in the answers. In this case, the resulting code will contain the Hadamard encoding of each of the possible answers provided by each of the servers; that is, if the j-th server answers according to $A_j(x,q) \in \{0,1\}^{\mathsf{ans}(k)}$, where $x \in \{0,1\}^k$ and $q \in \{0,1\}^{\mathsf{qst}(k)}$, then $\mathrm{C}(x)_{j,q,\ell} = \mathrm{C}_{\mathrm{Had}}(A_j(x,q))_\ell$, for every $\ell \in \{0,1\}^{\mathsf{ans}(k)}$. Thus, the length of the code is $s \cdot 2^{\mathsf{qst}(k)} \cdot 2^{\mathsf{ans}(k)}$. Now, on input $i \in [k]$, the decoder emulates the PIR user, obtaining the query sequence $(q_1,...,q_s)$ and the desired linear combinations $(\ell_1,....,\ell_s)$. It uniformly selects $r_1,...,r_s \in \{0,1\}^{\mathsf{ans}(k)}$, queries the (possibly corrupted) codeword at locations $(1,q_1,r_1),(1,q_1,r_1 \oplus \ell_1),...,(s,q_s,r_s),(s,q_s,r_s \oplus \ell_s)$, and XORs the $2s$ answers.

As mentioned above, better locally testable codes are known, but their construction is more involved (cf. [9]). Again, it is instructive to consider first the construction of PIR schemes, in which case s servers allow for a scheme with overall communication $k^{\epsilon(s)}$, where $\epsilon(s) = O(\log\log s)/s\log s \ll 1/(2s-1)$. In particular, $\epsilon(3) = 4/21$ improving over the previous bound of $1/5$.

Theorem 4.3 [9]: *For every constant q, there exist q-locally decodable binary codes of length $n = \exp(k^{\epsilon(q)})$, where $\epsilon(q) = \frac{O(\log\log q)}{q\log q}$.*

4.2.2 Polylog-local decoding for codes of nearly linear length

We will consider a code $\mathrm{C} : \Sigma^k \to \Sigma^n$ of linear distance, while identifying Σ with a finite field. For parameters h and $m = \log_h k$, consider a finite field F of size $O(m \cdot h)$, and a subset $H \subset F$ of size h. Viewing the information as a function $f : H^m \to F$, we encode it by providing the values of its low-degree extension $\widehat{f} : F^m \to F$ on all points in F, where $\widehat{f}$ is a m-variant polynomial of degree $|H|-1$ in each variable. Thus, the code maps $k = h^m$ long sequences over F (which may be viewed as $h^m \log|F|$ bits of information) to sequences of length $n \stackrel{\text{def}}{=} |F|^m = O(mh)^m = O(m)^m \cdot k$ over F. This code has relative distance $mh/|F|$. Note that the smaller m, the better the rate (i.e., relation of n to k) is, but this comes at the expense of using a larger alphabet F (as well as larger query complexity of the decoder presented below).

The decoder works by applying the self-correction paradigm. Given a point $x \in H^m$ and access to an oracle $w : F^m \to F$ that is 1/2-close to $\widehat{f}$, the value of $f(x)$ is recovered by uniformly selecting a line through x, querying for the $|F|$ values of w along the line, finding the degree mh univariant polynomial with the greatest agreement with these values, and evaluating it at the adequate point. Thus, we obtain an $|F|$-local decoder.

Using a constant m, we obtain an $O(k^{1/m})$-locally decodable code of constant rate (i.e., $n = O(k)$), over an alphabet of size $O(k^{1/m})$. On the other hand, using $m = \epsilon\log k/\log\log k$ (for any constant $\epsilon > 0$), we obtain a poly$(\log k)$-locally decodable code of length $n = k^{1+\epsilon}$, over an alphabet of size poly$(\log k)$. Concatenation with any reasonable[28] binary code (coupled with a trivial decoder that reads the entire codeword), yields a binary poly$(\log k)$-locally decodable code of length $n = k^{1+\epsilon}$.

[28] Indeed, we may use any good code (i.e., linear length and linear distance), as such can be easily constructed for block length $O(\log\log k)$. But we can even use the Hadamard code, because the length overhead caused by it in this setting is negligible.

4.2.3 Lower Bounds

It is known that locally decodable codes cannot be T2-nearly linear: Specifically, any q-locally decodable code $\mathrm{C} : \Sigma^k \to \Sigma^n$ must satisfy $n = \Omega(k^{1+\frac{1}{q-1}})$ (cf. [34]). For $q = 2$ and $\Sigma = \{0,1\}$, an exponential lower bound is known (cf. [35], following [31]). We conjecture that locally decodable codes cannot have polynomial length. In fact, we conjecture that locally decodable codes must have sub-exponential length.

Conjecture 4.4 *For every q there exists an $\epsilon > 0$ such that, for every $\delta > 0$ and all sufficienly large k, if $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ has a q-local δ-decoder then $n > \exp(k^\epsilon)$.*

We actually conjecture the same for families of codes over arbitrary alphabets, even when the alphabet size grows arbitrarily with k.

4.3 Relaxations

In light of the aforementioned conjecture it is natural to seek relaxations to the notion of locally decodable codes. One natural relaxation requires local recovery of most individual information-bits, allowing for recovery-failure (but not error) on the rest [15]: That is, it is requires that, whenever few location are corrupted, based on a constant number of queries, the decoder should be able to recover most of the individual information-bits, and for the rest of the locations, the decoder may output a fail symbol (but not the wrong value). Augmenting these requirements by the requirement that whenever the codeword is not corrupted – all bits are recovered correctly (with high probability), yields the following definition.

Definition 4.5 *For functions $n, \sigma : \mathsf{N} \to \mathsf{N}$, let $\mathrm{C} = \{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k \in K}$. For $q \in \mathsf{N}$ and $\delta, \rho \in (0,1)$, a* q-local relaxed (δ,ρ)-decoder for C *is a probabilistic* (non-adaptive) *oracle machine M that makes at most q queries and satisfies the following conditions:*

Local recovery from uncorrupted codewords: *For every $i \in [k]$ and $x = (x_1, ..., x_k) \in \Sigma^k$, it holds that $\Pr[M^{\mathrm{C}(x)}(1^k, i) = x_i] > 2/3$,*

Relaxed local recovery from somewhat corrupted codewords: *For every $x = (x_1, ..., x_k) \in \Sigma^k$, and any $w \in \Sigma^n$ that is δ-close to $\mathrm{C}(x)$, the following two conditions hold:*

1. *For every $i \in [k]$, it holds that $\Pr[M^{\mathrm{C}(x)}(1^k, i) \in \{x_i, \perp\}] > 2/3$, where $\perp$ is a special* ("failure") *symbol.*
2. *There exists a set $I_w \subseteq [k]$ of size at least ρk such that, for every $i \in I_w$, it holds that $\Pr[M^{\mathrm{C}(x)}(1^k, i) = x_i] > 2/3$.*[29]

In such a case, C *is said to be* locally relaxed-decodable.

It turns out (cf. [15]) that Condition 2, in the relaxed recovery requirement, essentially follows from the other requirements. That is, codes satisfying the other requirements can be transformed into locally relaxed-decodable codes, while essentially preserving their rate (and distance). Furthermore, the resulting codes satisfy the following stronger form of Condition 2: *There exists a set $I_w \subseteq [k]$ of density at least $1 - O(\Delta(w, \mathrm{C}(x))/n)$ such that for every $i \in I_w$ it holds that $\Pr[M^{\mathrm{C}(x)}(1^k, i) = x_i] > 2/3$.*

[29] We stress that it is not required that $\Pr[M^{\mathrm{C}(x)}(1^k, i) = \perp] > 2/3$ for $i \in [k] \setminus I_w$. Adding this requirement collapses the notion of relaxed-decodability to ordinary decodability (cf. [20]).

Theorem 4.6 [15]: *There exist locally relaxed-decodable codes of T1-nearly linear length. Specifically, for every* $\epsilon > 0$*, there exists codes of length* $n = k^{1+\epsilon}$ *that have a* $O(1/\epsilon^2)$*-local relaxed* $(\Omega(\epsilon), 1 - O(\epsilon))$*-decoder.*

An obvious open problem is to separate locally decodable codes from relaxed ones. This may follow by either improving the aforementioned lower bound on the length of locally decodable codes or by providing relaxed locally decodable codes of T2-nearly linear length.

References

[1] N. Alon, M. Krivelevich, T. Kaufman, S. Litsyn, and D. Ron. Testing low-degree polynomials over GF(2). In *Proceedings of the 7th RANDOM*, pages 188–199, 2003.

[2] A. Ambainis. An Upper Bound On The Communication Complexity of Private Information Retrieval. In *24th ICALP*, Springer, Lecture Notes in Computer Science, Vol. 1256, pages 401–407, 1997.

[3] S. Arora. *Probabilistic checking of proofs and the hardness of approximation problems.* PhD thesis, UC Berkeley, 1994.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM 45*, 3 (May 1998), 501–555. (Preliminary Version in *33rd FOCS*, 1992).

[5] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM 45*, 1 (Jan. 1998), 70–122. (Preliminary Version in *33rd FOCS*, 1992).

[6] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[7] L. Babai, L. Fortnow, L.A Levin and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symposium on the Theory of Computing*, May 1991, pp. 21–31.

[8] B. Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, Oct. 2001, pp. 106–115.

[9] A. Beimel, Y. Ishai, E. Kushilevitz, and J.F. Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *Proc. 43rd IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 261–270.

[10] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing in characteristic two. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 432–441, 1995.

[11] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal on Computing 27*, 3 (June 1998), 804–915. (Preliminary Version in *36th FOCS*, 1995).

[12] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proc. 25th ACM Symposium on the Theory of Computing*, May 1993, pp. 294–304.

[13] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 184–193, 1994.

[14] E. Ben-Sasson, O. Goldreich and M. Sudan. Bounds on 2-Query Codeword Testing. In the proceedings of *RANDOM'03*, Springer LNCS, Vol. 2764, pages 216–227, 2003.

[15] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs and Applications to Coding. In *Proc. 36th ACM Symposium on the Theory of Computing*, June 2004, pp. 1–10. See ECCC Technical Report TR04-021, March 2004.

[16] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova. Some 3CNF properties are hard to test. In *Proc. 35th ACM Symposium on the Theory of Computing*, June 2003, pp. 345–354.

[17] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. See ECCC Technical Report TR04-060, 2004.

[18] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proc. 35th ACM Symposium on the Theory of Computing*, June 2003, pp. 612–621.

[19] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Science 47*, 3 (Dec. 1993), 549–595. (Preliminary Version in *22nd STOC*, 1990).

[20] H. Buhrman and R. de Wolf. On relaxed locally decodable codes. Unpublished manuscript, July 2004.

[21] R. Canetti, O. Goldreich, S. and Halevi. The random oracle methodology, revisited. In *Proc. 30th ACM Symposium on the Theory of Computing*, May 1998, pp. 209–218.

[22] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, Private Information Retrieval. *Journal of the ACM*, Vol. 45, No. 6, pages 965–982, November 1998.

[23] I. Dinur and O. Reingold. PCP testers: Towards a more combinatorial proof of PCP theorem. To appear in *45th IEEE Symposium on Foundations of Computer Science*, 2004.

[24] F. Ergün, R. Kumar, and R. Rubinfeld. Fast approximate PCPs. In *Proc. 31st ACM Symposium on the Theory of Computing*, May 1999, pp. 41–50.

[25] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM 43*, 2 (Mar. 1996), 268–292. (Preliminary version in *32nd FOCS*, 1991).

[26] G.D. Forney. *Concatenated Codes.* MIT Press, Cambridge, MA 1966.

[27] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science 134*, 2 (Nov. 1994), 545–557.

[28] K. Friedl and M. Sudan. Some improvements to total degree tests. In *Proc. 3rd Israel Symposium on Theoretical and Computing Systems* (Tel Aviv, Israel, 4–6 Jan. 1995), pp. 190–198.

[29] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proc. 23rd ACM Symposium on the Theory of Computing*, pages 32–42, 1991.

[30] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM 45*, 4 (July 1998), 653–750. (Preliminary Version in *37th FOCS*, 1996).

[31] O. Goldreich, H. Karloff, L. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. In *Proc. 17th Conference on Computational Complexity* (Montréal, Québec, Canada, 21–24 May 2002), pp. 175–183.

[32] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost linear length. In *Proc. 43rd IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 13–22. (See ECCC Report TR02-050, 2002).

[33] P. Harsha and M. Sudan. Small PCPs with low query complexity. *Computational Complexity 9*, 3–4 (Dec. 2000), 157–201. (Preliminary Version in *18th STACS*, 2001).

[34] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd ACM Symposium on the Theory of Computing*, May 2000, pp. 80–86.

[35] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *Proc. 35th ACM Symposium on the Theory of Computing*, June 2003, pp. 106–115.

[36] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th ACM Symposium on the Theory of Computing*, May 1992, pp. 723–732.

[37] D. Lapidot and A. Shamir. Fully parallelized multi prover protocols for NEXP-time (extended abstract). In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, Oct. 1991, pp. 13–18.

[38] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992.

[39] S. Micali. Computationally sound proofs. *SIAM Journal on Computing 30*, 4 (2000), 1253–1298. (Preliminary Version in *35th FOCS*, 1994).

[40] A. Polishchuk and D.A. Spielman. Nearly-linear size holographic proofs. In *Proc. 26th ACM Symposium on the Theory of Computing*, May 1994, pp. 194–203.

[41] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing 25*, 2 (Apr. 1996), 252–271. (Preliminary Version in *3rd SODA*, 1992).

[42] 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. Also appears as Lecture Notes in Computer Science, Vol. 1001, Springer, 1996.

[43] M. Szegedy. Many-Valued Logics and Holographic Proofs. In *ICALP*, 1999, pages 676–686.