Lecture Notes on Locally Testable Codes and Proofs

Oded Goldreich*

June 3, 2016

Summary: We survey known results regarding locally testable codes and locally testable proofs (known as PCPs). Local 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., being a codeword or a valid proof).

In terms of property testing, locally testable codes are error correcting codes such that the property of being a codeword can be tested within low query complexity. As for locally testable proofs (PCPs), these can be viewed as massively parametrized properties that are testable within low query complexity such that the parameterized property is non-empty if and only if the corresponding parameter is in a predetermined set (of "valid statements").

Our first priority is minimizing the number of probes, and we focus on the case that this number is a constant. In this case (of a constant number of probes), we aim at minimizing the length of the constructs. That is, we seek locally testable codes and proofs of short length.

We stress a fundamental difference between the study of locally testable codes and the study of property testing. Locally testable codes are artificially designed with the aim of making codeword testing easy. In contrast, property testing envisions natural objects and properties that are prescribed by an external application.

This text has been adapted from our survey [33, 35], which was intended for readers having general background in the theory of computation. We chose to maintain this feature of the original text, and keep this text self-contained. Hence, the property testing perspective is mentioned but is not extensively relied upon. In particular, the fact that locally testable codes correspond to a special case of property testing is not pivotal to the presentation, although it is mentioned. Viewing PCPs in terms of property testing is less natural, yet this perspective is offered too (even in the foregoing summary); but again it is not pivotal to the presentation.

This text also differs from the other lecture notes in its style: It only provides *overviews* of results and proofs, rather than detailed proofs. Furthermore, the footnotes provide additional details that may be more essential than in other lectures.

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel.

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, we may want to recover the data and re-encode it whenever we find that the current encoding is corrupted. Doing so may allow to maintain the data integrity over eternity, although the encoded bits may all get corrupted in the course of time. Of course, we 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 we 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 probabilistic testing of whether a give string is a valid codeword or is far from any such 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 (i.e., distinguish, with high probability, between valid codewords and strings that are far from the code), but this would suffice in many applications (including the one outlined in the previous paragraph).

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 (i.e., the alleged proof). The tester's (a.k.a. verifier's) verdict is only correct with high probability, but this may suffice for many applications, where the assertion is rather mundane but of great practical importance. In particular, it suffices in applications in which proofs are used for establishing the correctness of *specific* computations of practical interest. 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 first priority is on minimizing the number of bits of the tested object that the tester reads, and we focus on the case that this number is a constant. In this case (of a constant number of probes), we aim at minimizing the length of these constructs (i.e., codes or proofs). An opposite regime, studied in [50, 51], refers to codes of linear length and seeks to minimize the number of bits read. We shall briefly review this alternative regime in Section 4.3.

Our interest in relatively *short* locally testable codes and proofs, is not surprising in view of the fact that *we envision such objects as actually being used in practice*. Of course, we do not suggest that one may actually 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 them 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.

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. When talking about oracle access to a string $w \in \{0,1\}^n$ we viewed w as a function $w : \{1,...,n\} \to \{0,1\}$. 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). Most importantly, this simplifies the composition of testers (see Section 3.2.1), and it comes at no real cost (since almost all known testers are actually non-adaptive).[1] Similarly, we focus on testers with one-sided error probability. Here, the main reason is aesthetic (since one-sided error is especially appealing in case of proof testers), and again almost all known testers are actually of this type.[2]

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 $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$, and the elements of $\{\mathcal{C}(x) : x \in \{0,1\}^k\} \subseteq \{0,1\}^n$ are called codewords (of $\mathcal{C}$).[3]

The distance of a code $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ is the minimum (Hamming) distance between its codewords; that is, $\min_{x \neq y}\{\Delta(\mathcal{C}(x), \mathcal{C}(y))\}$, 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 $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ of distance $\Omega(n)$.

The distance of $w \in \{0,1\}^n$ from a code $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$, denoted $\Delta_{\mathcal{C}}(w)$, is the minimum distance between w and the codewords of $\mathcal{C}$; that is, $\Delta_{\mathcal{C}}(w) \stackrel{\text{def}}{=} \min_x\{\Delta(w, \mathcal{C}(x))\}$. 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 $\mathcal{C}$ (resp., δ-close to $\mathcal{C}$) if $\Delta_{\mathcal{C}}(w) > \delta \cdot n$ (resp., $\Delta_{\mathcal{C}}(w) \leq \delta \cdot n$).

Loosely speaking, a codeword tester (or a tester for the code $\mathcal{C}$) is a tester for the property of being a codeword; that is, such a tester should accept any valid codeword, and reject (with high probability) any string that is far from being a codeword. In the following (basic) version of this notion, we fix the proximity parameter ϵ (which determines which words are considered "far" from the code) as well as the query complexity, denoted q. (Furthermore, since we consider a one-sided error version, we fix the rejection probability to $1/2$ (rather than to $1/3$).)[4]

Definition 1 (codeword tests, basic version): *Let $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ be a code* (of distance d)*, and let $q \in \mathbb{N}$ and $\epsilon \in (0,1)$. A q-*local (codeword) *ϵ-*tester for $\mathcal{C}$ *is a* (non-adaptive) *probabilistic*

[1] In particular, all testers that we shall present are non-adaptive. Furthermore, any oracle machine that makes a constant number of queries (to a binary oracle) can be emulated by a non-adaptive machine that makes a constant number of queries to the same oracle, albeit the second constant is exponential in the first one. Finally, we mention that if the code is linear, then adaptivity is of no advantage [15].

[2] In the context of proof testing (or verification), one-sided error probability is referred to as *perfect completeness*. We mention that in the context of linear codes, one-sided error testing comes with no extra cost [15].

[3] Indeed, we use $\mathcal{C}$ to denote both the encoding function (i.e., the mapping from k-bit strings to n-bit codewords) and the set of codewords.

[4] This is done in order to streamline this definition with the standard definition of PCP. As usual, the error probability can be decreased by repeated invocations of the tester.

oracle machine M that makes at most q queries and satisfies the following two conditions:[5]

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

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

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

The foregoing definition is interesting only in case ϵn is smaller than the covering radius of $\mathcal{C}$ (i.e., the smallest r such that for every $w \in \{0,1\}^n$ it holds that $\Delta_{\mathcal{C}}(w) \leq r$).[6] Actually, we shall focus on the case that $\epsilon < d/2n \leq r/n$, while noting that the case $\epsilon > 1.01d/n$ is of limited interest (see Exercise 1).[7] On the other hand, observe that $q = \Omega(1/\epsilon)$ must hold, which means that we focus on the case that $d = \Omega(n/q)$.

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

Definition 2 (codeword tests, asymptotic version): *For functions $n, d : \mathbb{N} \to \mathbb{N}$, let $\mathcal{C} = \{\mathcal{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k \in K}$ be such that $\mathcal{C}_k$ is a code of distance $d(k)$. For functions $q : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to (0,1)$, we say that a machine M is a q-*local (codeword) *ϵ-*tester for *$\mathcal{C} = \{\mathcal{C}_k\}_{k \in K}$ if, for every $k \in K$, machine M is a $q(k)$-local $\epsilon(k)$-tester for $\mathcal{C}_k$. Again, q is called the* query complexity *of M, and ϵ the* proximity parameter.

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

Definition 3 (locally testable codes (LTCs)): *Let n, d and $\mathcal{C}$ be as in Definition 2 and suppose that $d = \Omega(n)$. We say that $\mathcal{C}$ is* locally testable *if for every constant $\epsilon > 0$ there exist a constant q and a probabilistic polynomial-time oracle machine M such that M is a q-local ϵ-tester for $\mathcal{C}$.*

We will be concerned of the growth rate of n as a function of k, for locally testable codes $\mathcal{C} = \{\mathcal{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k \in K}$ of distance $d = \Omega(n)$. In other words, our main focus is on the case in which $\epsilon > 0$ and q are fixed constants. (More generally, for $d = \Omega(n)$, one may consider the trade-off between n, the proximity parameter ϵ, and the query complexity q.)

[5] In order to streamline this definition with the definition of PCP, we provide the tester with 1^k (rather than with n) as an explicit input. Since $n = n(k)$ can be determined based on k, the tester can determine the length of its oracle (before making any query to it). Recall that providing the tester with the length of its oracle is the standard convention in property testing.

[6] Note that $\lfloor d/2 \rfloor \leq r \leq n - \lceil d/2 \rceil$. The lower bound on r follows by considering a string that resides in the middle of the shortest path between two distinct codewords, and the upper bound follows by considering the distance of any string to an arbitrary set of two codewords. Codes satisfying $r = \lfloor d/2 \rfloor$ do exist but are quite pathologic (e.g., the code $\{0^t, 1^t\}$). The typical case is of $r \approx d$ (see, e.g., Hadamard codes and the guideline to Exercise 1).

[7] This observation was suggested to us by Zeev Dvir. Recall that the case of $\epsilon \geq r/n$, which implies $\epsilon \geq d/2n$, is always trivial.

2.2 Proof testers

We start by recalling the standard definition of PCP.[8] Here, the verifier is explicitly given a main input, denoted x, and is provided with oracle access to an alleged proof, denoted π; that is, the verifier can read x for free, but its access to π is via queries and the number of queries made by the verifier is the most important complexity measure. Another key complexity measure is the length of the alleged proof (as a function of $|x|$).

Definition 4 (PCP, standard definition): *A* probabilistically checkable proof (PCP) system *for a set S is a* (non-adaptive) *probabilistic polynomial-time oracle machine* (called a verifier)*, denoted V, satisfying*

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

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

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

Note that the proof complexity (i.e. p) of V is upper-bounded by $2^r \cdot q$, where r and q are the *randomness complexity* and the query complexity of the verifier, respectively. On the other hand, all known PCP constructions have randomness complexity that is at most logarithmic in their proof complexity (and in some sense this upper-bound always holds [9, Prop. 11.2]). Thus, the proof complexity of a PCP is typically captured by its randomness complexity, and the latter is prefered since using it is more convenient when composing proof systems (cf. Section 3.2.2).

recall that the proof complexity of V is defined as the number of bits in the proof that are inspected by V; that is, it is the "effective length" of the proof. Typically, this effective length equals the actual length; that is, all known PCP constructions can be easily modified such that the oracle locations accessed by V constitute a prefix of the oracle (i.e., $\bigcup_{\omega\in\{0,1\}^*} Q_x(\omega) = \{1,...,p(|x|)\}$ holds, for every x). (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 5 (PCP, augmented): *For functions $q : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to (0,1)$, we say that a PCP system V for a set S is a q-*locally ϵ-testable proof system *if it has query complexity q and satisfies the following condition, which augments the standard soundness condition.*[9]

[8] For a more paced introduction to the subject as well as a wider perspective, see [34, Chap. 9].

[9] Definition 5 relies on two natural conventions:

1. All strings in Π_x are of the same length, which equals $|\bigcup_{\omega\in\{0,1\}^*} Q_x(\omega)|$, where $Q_x(\omega)$ is as in Definition 4. Furthermore, we consider only π's of this length.
2. If $\Pi_x = \emptyset$ (which happens if and only if $x \notin S$), then every π is considered ϵ-far from Π_x.

These conventions will also be used in Definition 6.

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

The proof complexity of V *is defined as in Definition 4.*

At this point it is natural to refer to the verifier V as a proof tester. Note that Definition 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 and only if $x \in S$. (The reader is encouraged to think of Π_x as of a set of (redundant) proofs for an NP-proof system, although this is only the most appealing case.) Typically, one also requires the existence of a polynomial-time procedure that, on input a pair (x, π), determines whether or not $\pi \in \Pi_x$.[10] For simplicity we assume that, for some function $p : \mathbb{N} \to \mathbb{N}$ and every $x \in \{0,1\}^*$, it holds that $\Pi_x \subseteq \{0,1\}^{p(|x|)}$. The resulting definition follows.

Definition 6 (locally testable proofs): *Suppose that, for some function* $p : \mathbb{N} \to \mathbb{N}$ *and every* $x \in \{0,1\}^*$, *it holds that* $\Pi_x \subseteq \{0,1\}^{p(|x|)}$. *For functions* $q : \mathbb{N} \to \mathbb{N}$ *and* $\epsilon : \mathbb{N} \to (0,1)$, *we say that a* (non-adaptive) *probabilistic polynomial-time oracle machine* V *is a* q-locally ϵ-tester for proofs in $\Pi = \{\Pi_x\}_{x \in \{0,1\}^*}$ *if* V *has query complexity* q *and satisfies the following conditions:*

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

Rejecting invalid proofs:[11] *For every* $x \in \{0,1\}^*$ *and every* $\pi \in \{0,1\}^{p(|x|)}$ *that is* $\epsilon(|x|)$*-far from* Π_x, *on input* x *and oracle access to* π, *machine* V *rejects* x *with probability at least* $\frac{1}{2}$.

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

This notion of locally testable proofs is closely related to the notion of probabilistically checkable proofs (i.e., PCPs). 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 criterion of PCPs refers only to false assertions (i.e., x's such that $\Pi_x = \emptyset$). Still, all known PCP constructions actually satisfy the stronger definition.[12]

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

[11] Recall that if $\Pi_x = \emptyset$, then all strings are far from it. Also, since the length of the valid proofs for x is predetermined to be $p(|x|)$, there is no point to consider alleged proofs of different length. Finally, note that the current definition of the proof complexity of V is lower-bounded by the definition used in Definition 4.

[12] **Advanced comment:** In some cases this holds only under a weighted version of the Hamming distance, rather under the standard Hamming distance. Alternatively, these constructions can be easily modified to work under the standard Hamming distance.

Needless to say, the 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.

2.3 Ramifications and relation to property testing

We first comment about a few definitional choices made above. Firstly, we chose to focus on one-sided error testers; that is, we only consider testers that always accept valid objects (i.e., accept valid codewords (resp., valid proofs) with probability 1). In the current context, this is more appealing than allowing two-sided error probability, but the latter weaker notion is meaningful too. A second choice, which is a standard one, was to fix the error probability (i.e., the probability of accepting objects that are far from valid), rather than introducing yet another parameter. Needless to say, the error probability can be reduced by sequential invocations 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. Section 2.3.1). Next, we discuss the relation of local testability to property testing (cf. Section 2.3.2) and to PCPs of Proximity (cf. Section 2.3.3), while reviewing the latter notion. In Section 2.3.4, we discuss the motivation for the study of *short* local testable codes and proofs. Finally (in Section 2.3.5), we mention a weaker definition, which seems natural only in the context of codes.

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 ϵ. 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 3, by requiring the existence of a function $q : (0,1) \to \mathbb{N}$ and an oracle machine M such that, for every constant $\epsilon > 0$, all (sufficiently large) k and all $w \in \{0,1\}^{n(k)}$, the following conditions hold:

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

An analogous strengthening applies to Definition 6. A special case of interest is when $q(\epsilon) = O(1/\epsilon)$. In this case, it makes sense to ask whether or not an even stronger "uniformity" condition may hold. Like in Definitions 1 and 2 (resp., Definitions 5 and 6), the tester M will not be given the proximity parameter (and so its query complexity cannot depend on it), but we shall only require it to reject with probability proportional to the distance of the oracle from the relevant set. For example, we may strengthen Definition 3, by requiring the existence of an oracle machine M and a *constant* q such that for 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 $\mathcal{C}$, then $\mathbf{Pr}[M^w(1^k) = 1] = 1$.
3. If w is δ-far from $\{\mathcal{C}(x) : x \in \{0,1\}^k\}$, then $\mathbf{Pr}[M^w(1^k) = 1] < 1 - \Omega(\delta)$.

More generally, we may require the existence of a monotonically non-decreasing function ϱ such that inputs that are δ-far from the code are rejected with probability at least $\varrho(\delta)$ (rather than with probability at least $\Omega(\delta)$).

2.3.2 Relation to Property Testing

Locally testable codes (and their corresponding testers) are essentially special cases of property testing algorithms, as defined in [61, 37]. 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 the first strengthening outlined in Section 2.3.1, while the second strengthening outlined in Section 2.3.1 is related to the notion of proximity oblivious testing (cf. [40]). Specifically, using the language of property testing (see definitions in the first lecture), we have

Definition 7 (locally testable codes, property testing formulations): *Let n, d and $\mathcal{C} = \{\mathcal{C}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}_{k \in K}$ be as in Definition 2, and suppose that $d = \Omega(n)$.*

1. Weak version:[13] *For $q : \mathbb{N} \times (0,1] \to \mathbb{N}$, we say that $\mathcal{C}$ is* uniformly q-locally testable *if there exists a non-adaptive tester of query complexity q and one-sided error for the property $\mathcal{C}$.*

   A special case of interest is when $q(k, \epsilon) = q'(\epsilon)$ for some function $q' : (0,1] \to \mathbb{N}$.

2. Strong version:[14] *We say that $\mathcal{C}$ is* locally testable in a ϱ-strong sense *if there exists a* (non-adaptive) *proximity oblivious tester of constant query complexity, detection probability ϱ, and one-sided error for the property $\mathcal{C}$.*

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 objects designed artificially to withstand noise (i.e., codewords of error correcting codes).

Our general formulation of proof testing (i.e., Definition 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 6 allows to consider properties that are parameterized by auxiliary information (i.e., x), which falls into the framework of massively parameterized properties (cf. [58]). Note that $\Pi_x \subseteq \{0,1\}^{p(|x|)}$, where p is typically a polymonial (which means that the length of the tested object is polynomial in the length of the parameter). In contrast, most property testing research refers to the case that

[13] Indeed, this version corresponds to the first strengthening outlined in Section 2.3.1. Note that the current formulation includes the case in which for every $\epsilon > 0$ there exists a $k_\epsilon \in \mathbb{N}$ such that $q(k, \epsilon) = q'(\epsilon)$ if $k \geq k_\epsilon$ and $q(k, \epsilon) = n(k)$ otherwise.

[14] Indeed, this version corresponds to the second strengthening outlined in Section 2.3.1. Recall that the restricted version of this definition referred to the case that ϱ is linear (i.e., $\varrho(\delta) = \Omega(\delta)$).

the length of the tested object is exponential in the length of the parameter (i.e., $\Pi_n \subseteq \{0,1\}^n = \{0,1\}^{\exp(|n|)}$).[15] Hence, using the language of property testing, we can reformulate Definition 6, as follows:

Definition 8 (locally testable proofs as property testers):[16] *Suppose that, for some function $p : \mathbb{N} \to \mathbb{N}$ and every $x \in \{0,1\}^*$, it holds that $\Pi_x \subseteq \{0,1\}^{p(|x|)}$, and let $q : \{0,1\}^* \times (0,1] \to \mathbb{N}$. We say that a* (non-adaptive) *probabilistic polynomial-time oracle machine V is a q-*locally tester for proofs in $\{\Pi_x\}_{x\in\{0,1\}^*}$ *if V has query complexity q and constitutes an tester for the parameterized property $\{\Pi_x\}_{x\in\{0,1\}^*}$, where such a tester gets x and ϵ as input parameters and ϵ-tests membership in Π_x using $q(x,\epsilon)$ queries.*

A special case of interest is when $q(x,\epsilon) = q'(\epsilon)$ for some function $q' : (0,1] \to \mathbb{N}$.

2.3.3 Relation to PCPs of Proximity

We start by reviewing the definition of a PCP of Proximity, which may be viewed as a "PCP version" of a property tester (or a "property testing analogue" of PCP).[17] In the following definition, the tester (or verifier) is given oracle access both to its main input, denoted x, and to an alleged proof, denoted π, and the query complexity account for its access to both oracles (which can be viewed as a single oracle, (x,π)). That is, in contrast to the definition of PCP and like in the definition of property testing, the main input is presented as an oracle and the verifier is charged for accessing it. In addition, like in the definition of PCP (and unlike in the definition of property testing), the verifier gets oracle access to an alleged proof.

Definition 9 (PCPs of Proximity):[18] *A* PCP of Proximity for a set S with proximity parameter ϵ *is a* (non-adaptive) *probabilistic polynomial-time oracle machine, denoted V, satisfying*

Completeness: *For every $x \in S$ there exists a string $\pi_x \in \{0,1\}^{p(|x|)}$ such that V always accepts when given access to the oracle (x,π_x); that is, $\mathbf{Pr}[V^{x,\pi_x}(1^{|x|})=1] = 1$.*

Soundness: *For every x that is ϵ-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, $\mathbf{Pr}[M^{x,\pi}(1^{|x|})=1] \leq 1/2$.*

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

The definition of a property tester (i.e., an ϵ-tester for S) is obtained as a special case by requiring that the proof length equals zero. Note that for any (efficiently computable) code C of constant

[15] Indeed, in the context of property testing, the length of the oracle must always be given to the tester (although some sources neglect to account for this fact).

[16] Here, we allow the query complexity to depends (also) on the parameter x, rather than merely on its length (which also determines the length $p(|x|)$ of the tested object). This seems more natural in the context of testing massively parameterized properties.

[17] An "NP version" (or rather an "MA version") of a property tester was presented by Gur and Rothblum [43]. In their model, called MAP, the verifier has oracle access to the main input x, but gets free access to an alleged proof π.

[18] Note that this definition builds on Definition 4 (rather than on Definition 6), except that the proof complexity is defined as in Definition 6. (We mention that PCPs of Proximity, introduced by Ben-Sasson *et al.* [11], are almost identical to Assignment Testers, introduced independently by Dinur and Reingold [24]. Both notions are (important) special cases of the general definition of a "PCP spot-checker" formulated before by Ergün *et al.* [26].)

relative distance, a PCP for a set S can be obtained from a PCP of Proximity for $\{C(x) : x \in S\}$, where the complexity of the PCP is related to the complexity of the PCP of Proximity via the rate of the code (since the complexities of proof testers are measured in terms of the length of their main input).[19]

Relation to locally testable proofs (a bit contrived). The definition of a PCP of Proximity is related to but different from the definition of a locally testable proof: Definition 9 refers to the distance of the input-oracle x from S, whereas locally testable proofs (see Definition 6) 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 proofs, by considering an artificial set of proofs for membership in a generic set. Specifically, given a PCP of Proximity verifier V of proof complexity p and proximity parameter ϵ for a set S, we consider the set of proofs (for membership of 1^n in the generic set $\{1\}^*$)

$$\Pi'_{1^n} \stackrel{\text{def}}{=} \left\{ x^t\pi : x \in (S \cap \{0,1\}^n) \ , \ \pi \in \Pi_x \ , \ t = \frac{p(n)}{\epsilon n} \right\} \tag{1}$$

$$\text{where } \Pi_x \stackrel{\text{def}}{=} \{\pi' \in \{0,1\}^{p(|x|)} : \mathbf{Pr}[V^{x,\pi'}(1^n) = 1]\} \tag{2}$$

so that $|\pi| = \epsilon \cdot |x^t|$. A 3ϵ-tester for proofs in $\Pi' = \{\Pi'_{1^n}\}_{n\in\mathbb{N}}$ can be obtained by emulating the execution of V and checking that the t copies in the tn-bit long prefix of the oracle are indeed identical.[20]

Digest: the use of repetitions. The problem we faced in the construction of Π' is that the proof-part (i.e., π) is essential for verification, but we wish the distance to be determined by the input-part (i.e., x). The solution was to repeat x multiple times so that these repetitions dominate the length of the oracle. The new tester can still access the alleged proof π, but we are guaranteed that if $x^t\pi$ is 3ϵ-far from Π', then x is 2ϵ-far from S. The repetition test is used in order to handle the possibility that the oracle does not have the form $x^t\pi$ but is rather ϵ-far from any string having this form.

PCPs of Proximity yield locally testable codes. 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 $\mathcal{C}_0$ with any PCP of Proximity V, and obtain a q-locally testable code with distance essentially determined by $\mathcal{C}_0$ and rate determined by V, where q is the query complexity of V. Specifically, x will be encoded by appending $c = \mathcal{C}_0(x)$ with a proof that c is a codeword of $\mathcal{C}_0$, and distances will be determined by the *weighted Hamming distance* that assigns all weight (uniformly) to the first part of the new code. As in the previous paragraph, these weights can be (approximately) "emulated" by making suitable repetitions. Specifically, the new codeword, denoted $\mathcal{C}(x)$, equals $\mathcal{C}_0(x)^t\pi(x)$, where $\pi(x)$ is the foregoing proof and $t = \omega(|\pi(x)|)/|\mathcal{C}_0(x)|$, and the new tester checks that the $t \cdot |\mathcal{C}_0(x)|$-bit long prefix contains t repetitions of some string and

[19] For details, see Exercise 2.

[20] That is, on input $x^{(1)} \cdots x^{(t)}\pi$, the verifier invokes $V^{x^{(1)},\pi}(1^n)$ as well as performs checks to verify that $x^{(1)} = x^{(i)}$ for every $i \in [t]$. The latter test is conducted by selecting uniformly several $i \in [t]$ and several $j \in [n]$ per each i, and comparing $x^{(1)}_j$ to $x^{(i)}_j$. The key observation is that if V is an ϵ-tester, and $x^{(1)} \cdots x^{(t)}\pi$ is 3ϵ-far from Π'_{1^n}, then either $x^{(1)} \cdots x^{(t)}$ is ϵ-far from $(x^{(1)})^t$ or $x^{(1)}$ is ϵ-far from S, since $|\pi| < \epsilon \cdot |x^{(1)} \cdots x^{(t)}\pi|$. See related Exercise 3.

invokes the PCP of Proximity while proving it access to a random copy (as main input) and to the $|\pi(x)|$-bit long suffix (as an alleged proof). See Exercise 3.

We stress that this only yields a weak locally testable code (e.g., as in Definition 3), since nothing is guaranteed for non-codewords that consists of repetitions of some $\mathcal{C}_0(x)$ and a false proof.[21] Obtaining a strong locally testable code using this method is possible when the PCP of Proximity is stronger in a sense that is analogous to Definition 6, but with a single valid proof (called canonical) per each $x \in S$ (i.e., $|\Pi_x| = 1$ for every $x \in S$). Such strong PCPs of Proximity were introduced in [41]; see also [38].

2.3.4 Motivation for 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) can often be done in time that is poly-logarithmic in the length of the probed object. Recall that the tested object is supposed to be related to some primal object; in the case of codes, the probed object is supposed to encode the primal object, whereas in the case of proofs the probed object is supposed to help verify some property of the primal object. In both cases, the length of the secondary (probed) object is of natural concern, and this length is stated in terms of the length of the primary object.

The length of codewords in an error-correcting code is widely recognized as one of the two most fundamental parameters of the code (the second one being the code's 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 Section 1, the same considerations apply also to proofs. However, in the case of proofs, this obvious point has been blurred by the unexpected and highly influential applications of PCPs to establishing hardness results regarding the complexity of natural approximation problems. 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 [49, 56] and their applications [7, 20]), and the length of the PCP affects the complexity of those applications.

Turning back to the celebrated application of PCP to the study of the complexity of natural approximation problems, we note that the length of PCPs is relevant also to these non-approximability results; specifically, the length of PCPs affects the *tightness with respect to the running time* of the non-approximability results derived from these PCPs. For example, suppose that (exact) SAT has complexity $2^{\Omega(n)}$. Then, while the original PCP Theorem [4, 3] only implies that approximating MaxSAT requires time 2^{n^α}, for some (small constant) $\alpha > 0$, the results of [17, 22] yield a lower bound of $2^{n/\mathrm{poly}(\log n)}$. We mention that the result of [57] (cf. [23]) allows to achieve a time lower

[21] **Advanced comment:** Actually, we can also achieve the special case of the weak version of Definition 7, where the query complexity only depends on the proximity parameter ϵ. This can be done by using $t = T(|x|) \cdot |\pi(x)|/|\mathcal{C}_0(x)|$ (e.g., $T(k) = \log k$) and claiming query complexity of the form $n(T^{-1}(O(1/\epsilon)))$ (e.g., $\exp(O(1/\epsilon))$, resp., assuming $n(k) \stackrel{\mathrm{def}}{=} |\mathcal{C}(1^k)| = \mathrm{poly}(k)$). The claim is valid beacuse the foregoing difficulty arises only when $\epsilon < 3/t$ or so, but in this case $n(T^{-1}(O(1/\epsilon)))) = n(|x|) = |\mathcal{C}(x)|$.

bound of $2^{n^{1-o(1)}}$ simultaneously with optimal non-approximability ratios, but this is currently unknown for the better lower bound of $2^{n/\mathrm{poly}(\log n)}$.

2.3.5 A weaker definition

One of the concrete motivations for locally testable codes refers to settings in which one may want to re-encode the information when discovering that the codeword is corrupted. In such a case, assuming that re-encoding is based solely on the corrupted codeword, one may assume (or rather needs to assume) that the corrupted codeword is not too far from the code. Thus, the following version of Definition 1 may suffice for various applications.

Definition 10 (weak codeword tests): *Let $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ be a code of distance d, and let $q \in \mathbb{N}$ and $\epsilon_1, \epsilon_2 \in (0,1)$ be such that $\epsilon_1 < \epsilon_2$. A* weak q-local (codeword) (ϵ_1, ϵ_2)-tester for $\mathcal{C}$ *is a* (non-adaptive) *probabilistic oracle machine M that makes at most q queries, accepts any codeword with probability 1, and rejects* (w.h.p.) *non-codewords that are both ϵ_1-far and ϵ_2-close to $\mathcal{C}$. That is, the rejection condition of Definition 1 is modified as follows.*

Rejection of non-codeword (weak version): *For any $w \in \{0,1\}^n$ such that $\Delta_{\mathcal{C}}(w) \in [\epsilon_1 n, \epsilon_2 n]$, given oracle access to w, machine M rejects with probability at least $1/2$.*

Needless to say, there is something highly non-intuitive in this definition: It requires rejection of non-codewords that are somewhat far from the code, but not the rejection of codewords that are very far from the code. Still, such weak codeword testers may suffice in some applications. Interestingly, such weak codeword testers seem easier to consrtruct than standard locally testable codes; they even achieve linear length (cf. [63, Chap. 5]), whereas this is not known for the standard notion (see Problem 12). We note that the non-monotonicity of the rejection probability of testers has been observed before; the most famous example being linearity testing (cf. [19] and [8]).

2.4 On relating locally testable codes and proofs

This section presents an advanced discussion, which is mainly intended for PCP enthusiasts. We discuss the common beliefs that locally testable codes and proofs are closely related, and point out that the relation is less clear than one may think.

Locally testable codes can be thought of as the combinatorial counterparts of the complexity theoretic notion of locally testable proofs (PCPs). In particular, as detailed below, the use of codes with features related to local testability is implicit in known PCP constructions. This perspective raises the question of whether one of these notions implies the other, or at least is useful towards the understanding of the other.

2.4.1 Do PCPs imply locally testable codes?

As started above, 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

$\mathbf{Pr}[V^\pi(x) = 1] = 1$, 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 corresponding PCP verifier yields a codeword tester, but this is not really the case.

The point is that Definition 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).[22] 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 5 is a strict superset of R_x (rather than $\Pi_x = R_x$). We comment that most known PCP constructions can be modified to yield $\Pi_x = R_x$, and thus to yield a locally testable code, but these modifications are far from being trivial. The interested reader is referred to [41, Sec. 5.2] for a discussion of typical problems that arise when trying this way. In any case, this is not necessarily the best way to obtain locally testable codes from PCPs; an alternative way is outlined in Section 2.3.3.

2.4.2 Do locally testable codes imply 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. [41, 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 length $\ell(k) = k^c$ for any constant $c > 1$. Unfortunately, their stronger result, providing a locally testable code of shorter length (i.e., length $\ell(k) = k^{1+o(1)}$) is obtained by constructing (cf. [41, Sec. 4]) and using (cf. [41, Sec. 5]) a corresponding locally testable proof (i.e., PCP).

Most subsequent works (e.g., [11, 22]) have followed this route (i.e., of going from a PCP to a code), but there are notable exceptions. Most importantly, we note that Meir's work [54] provides a combinatorial construction of a locally testable code that does not seem to yield a corresponding locally testable proof. The prior work of Ben-Sasson and Sudan [17] may be viewed as reversing the course to the "right one": They first construct locally testable codes, and next use them towards the construction of proofs, but their set of valid codewords is an NP-complete set. Still, conceptually they go from codes to proofs (rather than the other way around).

3 Results and Ideas

We review some of the the known 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. In all cases, we refer to testers of constant query complexity.[23] Before embarking on this journey, we mention that random linear codes (of linear length) require any codeword tester to read a linear number of bits of the codeword (see exercises in the first lecture). Furthermore, good codes that

[22] Let alone that Definition 4 refers only to the case of false assertions, in which case all strings are far from any valid proof (since the latter does not exist).

[23] The opposite regime, in which the focus is on linear length codes and the aim is to minimize the query complexity, is briefly reviewed in Section 4.3.

correspond to random "low density parity check" matrices are also as hard to test [15]. These facts provide a strong indication to the non-triviality of local testability.

Teaching note: Recall that this section only provides overviews of the constructions and their analysis. The intention is merely to provide a taste of the ideas used. The interested reader should look for detailed descriptions in other sources, which are indicated in the text.

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 of the form $\ell(z) = \sum_{i=1}^{k} \ell_i z_i$ 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 amounts 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). Hence, local testability of $\mathrm{C}_{\mathrm{Had}}$ is achieved by invoking the linearity tester of Blum, Luby, and Rubinfeld [19], which amounts to uniformly selecting $y, z \in \{0,1\}^k$ and checking whether $w(y+z) = w(y) + w(z)$.

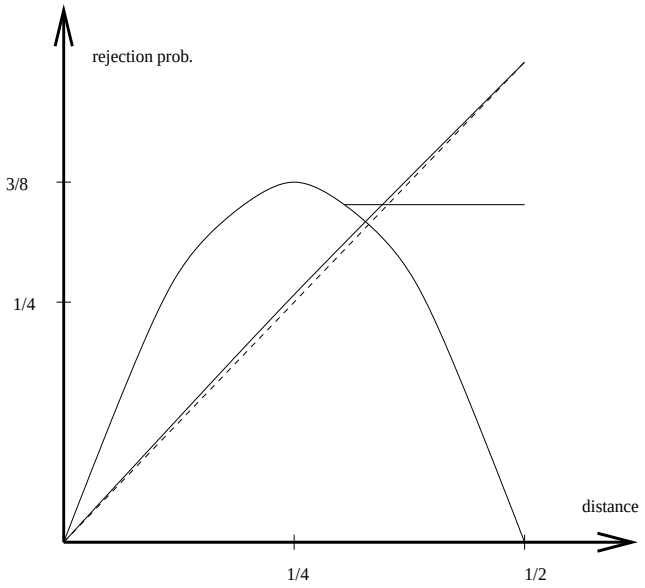

Figure 1: The lower bounds that underlie the function Γ. The dashed diagonal line represents the bound $\varrho(x) \geq x$, which is slightly improved by the bound $\varrho(x) \geq x + \eta(x)$.

This natural tester always accepts linear functions, and (as shown in the lecture on linearity testing) reject any function that is δ-far from being linear with probability at least $\min(\delta/2, 1/6)$.

Recall that the exact behavior of this tester is unknown; that is, denoting by $\varrho(\delta)$ the minimum rejection probability of a string that is at (relative) distance δ from $\mathrm{C}_{\mathrm{Had}}$, we know lower and upper bounds on ϱ that are tight only in the interval $[0, 5/16]$ (and at the point 0.5). Specifically, it is known that $\varrho(\delta) \geq \Gamma(\delta)$, where the function $\Gamma : [0, 0.5] \to [0, 1]$ is defined as follows:

$$\Gamma(x) \stackrel{\text{def}}{=} \begin{cases} 3x - 6x^2 & 0 \leq x \leq 5/16 \\ 45/128 & 5/16 \leq x \leq \tau_2 \text{ where } \tau_2 \approx 44.9962/128 \\ x + \eta(x) & \tau_2 \leq x \leq 1/2, \\ & \text{where } \eta(x) \stackrel{\text{def}}{=} 1376 \cdot x^3 \cdot (1-2x)^{12} \geq 0. \end{cases} \tag{3}$$

The lower bound Γ is composed of three different bounds with "phase transitions" at $x = \frac{5}{16}$ and at $x = \tau_2$, where $\tau_2 \approx \frac{44,9962}{128}$ is the solution to $x + \eta(x) = 45/128$ (see Figure 1).[24] It was shown in [8] that the first segment of Γ (i.e., for $x \in [0, 5/16]$) is the best bound possible, and that the first "phase transitions" (i.e., at $x = \frac{5}{16}$) is indeed a reality; in other words, $\varrho = \Gamma$ in the interval $[0, 5/16]$.[25] We highlight the non-trivial behavior of the detection probability of the aforementioned test, and specifically the fact that the detection probability does not increase monotonically with the distance of the tested string from the code (i.e., Γ decreases in the interval $[1/4, 5/16]$, while equaling ϱ in this interval).

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 [9], but this code has double-exponential length (and was introduced merely for the design of PCPs).[26]

3.1.2 The Hadamard-Based PCP of ALMSS

The simplest example of a locally testable proof (for arbitrary sets in $\mathcal{NP}$)[27] is the "inner verifier" of the PCP construction of Arora, Lund, Motwani, Sudan and Szegedy [3], which in turn is based on the Hadamard code. Specifically, proofs of the satisfiability of a given system of quadratic equations over GF(2), which is an NP-complete problem (see Exercise 5), are presented by providing a Hadamard encoding of the outer-product of a satisfying assignment with itself (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 $\beta_{i,j} = \alpha_i \alpha_j$). Hence, the alleged proofs are of length 2^{n^2}, and locations in these proofs correspond to n^2-bit long strings (or, equivalently, to n-by-n Boolean matrices).

Given an alleged proof $\pi \in \{0,1\}^{2^{n^2}}$, viewed as a Boolean function $\pi : \{0,1\}^{n^2} \to \{0,1\}$, the proof-tester (or verifier) proceeds as follows:[28]

[24] The third segment is due to [47], which improves over the prior bound of [8] that asserted $\varrho(x) \geq \max(45/128, x)$ for every $x \in [5/16, 1/2]$.

[25] In contrast, the lower bound provided by the other two segments (i.e., for $x \in [5/16, 1/2]$) is unlikely to be tight, and in particular it is unlikely that the "phase transitions" at $x = \tau_2$ represents the behavior of ϱ itself. We also note that $\eta(x) > 59 \cdot (1-2x)^{12}$ for every $x > \tau_2$, but $\eta(x) < 0.0001$ for every $x < 1/2$.

[26] Interestingly, some of the best PCP results are obtained by using a *relaxed* notion of local testability [44, 45].

[27] A simpler example for a set not known to be in $\mathcal{BPP}$ is provided by the interactive proof for graph non-isomorpishm [39]. Note that any interactive proof system in which the prover sends a constant number of bits yields a PCP system (see Exercise 4).

[28] See [34, Sec. 9.3.2.1] for a more detailed description.

1. Tests that π is indeed a codeword of the Hadamard Code (i.e., that it is a linear function from $\{0,1\}^{n^2}$ to $\{0,1\}$). If this test passes (with high probability), then π is close to some codeword $\mathrm{C}_{\mathrm{Had}}(\beta)$, for an arbitrary $\beta = (\beta_{i,j})_{i,j\in[n]}$; that is, for (say) 99% of the Boolean matrices $C = (c_{i,j})_{i,j\in[n]}$, it holds that $\pi(C) = \sum_{i,j\in[n]} c_{i,j}\beta_{i,j}$.

2. Tests that the aforementioned β is indeed an outer-product of some $\alpha \in \{0,1\}^n$ with itself. This means that for every $C = (c_{i,j})_{i,j\in[n]}$ (or actually for 99% of them), it holds that $\pi(C) = \sum_{i,j\in[n]} c_{i,j}\alpha_i\alpha_j$. That is, 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 Boolean matrices).

   Teaching note: Some readers may prefer to skip the description of how the current step is implemented, proceed to Step 3, and return to the current step later.

   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 $\sum_{i=1}^n c_i\beta_{i,i}$).[29] So we would like to test that the latter codeword matches the former one. (Recall that this means testing whether the matrix $(\beta_{i,j})_{i,j\in[n]}$ equals the matrix $(\alpha_i\alpha_j)_{i,j\in[n]}$.)

   This test can be performed by uniformly selecting $(r_1,...,r_n),(s_1,...,s_n) \in \{0,1\}^n$, and comparing $\sum_{i,j} r_is_j\beta_{i,j}$ and $\sum_{i,j} r_is_j\alpha_i\alpha_j = (\sum_i r_i\alpha_i)\cdot(\sum_j s_j\alpha_j)$, where the value $\sum_{i,j} r_is_j\beta_{i,j}$ is supposed to reside in the location that corresponds to the outer-product of $(r_1,...,r_n)$ and $(s_1,...,s_n)$. The key observation here is that for n-by-n matrices $A \neq B$, when $r,s \in \{0,1\}^n$ are uniformly selected (vectors), it holds that $\mathbf{Pr}_s[As = Bs] = 2^{-\mathrm{rank}(A-B)}$ and it follows that $\mathbf{Pr}_{r,s}[rAs = rBs] \leq 3/4$ (see Exercises 6).

   The foregoing suggestion would have been fine if $\pi = \mathrm{C}_{\mathrm{Had}}(\beta)$, but we only know that π 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).[30] 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 $\sum_{i,j\in[n]} c_{i,j}\beta_{i,j}$, we uniformly select $\omega = (\omega_{i,j})_{i,j\in[n]}$ and retrieve both $\pi(\omega)$ and $\pi(\omega + c)$. Thus, we obtain a self-corrected value of $\pi(c)$; that is, if π is δ-close to $\mathrm{C}_{\mathrm{Had}}(\beta)$ then $\pi(\omega + c) - \pi(\omega) = \sum_{i,j\in[n]} c_{i,j}\beta_{i,j}$ with probability at least $1 - 2\delta$ (over the choice of ω).

   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 $\sum_{i,j} r_is_j\beta_{i,j}$ and $(\sum_i r_i\beta_{i,i})\cdot(\sum_j s_j\beta_{j,j})$, for randomly selected $(r_1,...,r_n),(s_1,...,s_n) \in \{0,1\}^n$.

3. Finally, we get to the purpose of all of the foregoing, which is checking whether the aforementioned α satisfies the given system of quadratic equations. Towards this end, the tester uniformly selects a linear combination of the equations, and checks whether α satisfies the (single) resulting equation. Note that the value of the corresponding quadratic expression (which

[29] Note that $\sum_{i\in[n]} c_i\beta_{i,i} = \sum_{i,j\in[n]} c_{i,j}\beta_{i,j}$, where $c_{i,j} = c_i$ if $i = j$ and $c_{i,j} = 0$ otherwise. Hence, the value of location $(c_1,...,c_n)$ in $\mathrm{C}_{\mathrm{Had}}(\alpha)$ appears at location $(c_{i,j})_{i,j\in[n]}$ in $\mathrm{C}_{\mathrm{Had}}(\beta)$.

[30] Likewise, the values at the locations that correspond the outer-product of $(r_1,...,r_n)$ and $(s_1,...,s_n)$ should not be retrieved directly, because these locations are a tiny fraction of all 2^{n^2} locations in $\mathrm{C}_{\mathrm{Had}}(\beta)$.

is a linear combination of quadratic (and linear) forms) appears as a bit of the Hadamard encoding of β, but again we retrieve it from π by using self-correction.

The foregoing description presumes that each step conducts a constant number of checks such that if the corresponding condition fails then this step rejects with high (constant) probability.[31] In the analysis, one shows that if π is 0.01-far from a valid Hadamard codeword, then Step 1 rejects with high probability. Otherwise, if π is 0.01-close to $\mathrm{C}_{\mathrm{Had}}(\beta)$ for $\beta = (\beta_{i,j})_{i,j\in[n]}$ that is *not* an outer-product of some $\alpha = (\alpha_i)_{i\in[n]}$ with itself (i.e., $(\beta_{i,j})_{i,j\in[n]} \neq (\alpha_i\alpha_j)_{i,j\in[n]}$), then Step 2 rejects with high probability. Lastly, if π is 0.01-close to $\mathrm{C}_{\mathrm{Had}}(\beta)$ such that $\beta_{i,j} = \alpha_i\alpha_j$ for some α (and all $i, j \in [n]$) but α does not satisfy the given system of quadratic equations, then Step 3 rejects with high probability.

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 much more challenging.

3.2.1 Locally testable codes of quadratic length

A rather natural interpretation of *low-degree tests* (cf. [5, 6, 31, 61, 30]) 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. [41]). We sketch both constructions below, starting with locally testable codes over very large alphabets (which are defined analogously to the binary case).

Locally testable codes over large alphabets. Recall that we presented low-degree tests for degree $d \ll |\mathcal{F}|$ and functions $f : \mathcal{F}^m \to \mathcal{F}$ as picking $d+2$ points over a random line (in $\mathcal{F}^m$) and checking whether the values of f on these points fits a degree d univariate polynomial. We also commented that such a test can be viewed as a PCP of Proximity that test whether f is of degree d by utilizing a proof-oracle (called a *line oracle*) that provides the univariate degree d polynomials that describe the value of f on every line in $\mathcal{F}^m$.[32] (When queried on $(\overline{x}, \overline{h}) \in \mathcal{F}^m \times \mathcal{F}^m$, this proof-oracle returns the $d+1$ coefficients of a polynomial that supposedly describes the value of f on the line $\{\overline{x} + i\overline{h} : i \in \mathcal{F}\}$, and the verifier checks that the value assigned by this polynomial to a random $i \in \mathcal{F}$ matches $f(\overline{x} + i\overline{h})$.)

Taking another step, we note that given access only to a "line oracle" $L : \mathcal{F}^m \times \mathcal{F}^m \to \mathcal{F}^{d+1}$, we can test whether L describes the restrictions of a single degree d multivariate polynomial to all lines. This is done by selecting a random pair of intersecting lines and checking whether they agree on the point of intersection. Friedl and Sudan [30] and Rubinfeld and Sudan [61] proposed to view each valid L as a codeword in a locally testable code over the alphabet $\Sigma = \mathcal{F}^{d+1}$. This

[31] An alternative description may have each step repeat the corresponding check only once so that if the corresponding condition fails, then this step rejects with some (constant) positive probability. In this case, the analysis will only establish that the entire test rejects with some (constant) positive probability, and repetitions will be used to reduce the soundness error to 1/2.

[32] This comment appears as a footnote in the last sectyion of the lecture notes on low degree testing. Recall that PCPs of Proximity were defined in Section 2.3.3.

code maps each m-variate polynomial of degree d to the sequence of univariate polynomials that describe the restrictions of this polynomial to all possible lines; that is, the polynomial p is mapped to $L_p : \mathcal{F}^m \times \mathcal{F}^m \to \mathcal{F}^{d+1}$ such that, for every $(\overline{x}, \overline{h}) \in \mathcal{F}^m \times \mathcal{F}^m$, it holds that $L_p(\overline{x}, \overline{h})$ is (or represents) a univariate polynomial that describes the value of p on the line $\{\overline{x} + i\overline{h} : i \in \mathcal{F}\}$. The corresponding 2-query tester of $L : \mathcal{F}^m \times \mathcal{F}^m \to \mathcal{F}^{d+1}$, will just select a random pair of intersecting lines and check whether they agree on the point of intersection.[33] The analysis of this tester reduces to the analysis of the corresponding low degree test, undertaken in [3, 59].

The question at this point is what are the parameters of the foregoing code, denoted $\mathcal{C} : \Sigma^k \to \Sigma^n$, where $\Sigma = \mathcal{F}^{d+1}$ (and $n = |\mathcal{F}^m|^2$).[34] This code has distance $(1-d/|\mathcal{F}|)\cdot n = \Omega(n)$, since different polynomials agree with probability at most $d/|\mathcal{F}|$ on a random point (and ditto on a random line). Since Σ^k corresponds to all possible m-variate polynomials of degree d over $\mathcal{F}$ (which have $\binom{m+d}{d}$ possible monomials), it follows that $\Sigma^k = |\mathcal{F}|^{\binom{m+d}{d}}$, which implies

$$k = \frac{\binom{m+d}{d}}{d+1} \approx \frac{(d/m)^m}{d} = \frac{d^{m-1}}{m^m}, \tag{4}$$

when $m \ll d$ (which is the preferred setting here (see next)). Note that $n = |\mathcal{F}^m|^2$, which means that $n \approx d^2 \cdot (m \cdot |\mathcal{F}|/d)^{2m} \cdot k^2 \gg k^2$, since $|\mathcal{F}| > d$. Lastly, $|\Sigma| = |\mathcal{F}|^{d+1} > k^{(d+1)/(m-1)} \gg k$. Hence, the smaller m, the better the rate (i.e., relation of n to k), but this comes at the expense of using a relatively larger alphabet. In particular, we consider two instantiations, where in both $|\mathcal{F}| = \Theta(d)$:

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

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

In both cases, we obtain a locally testable code of polynomial length, but this code uses a large alphabet, whereas we seek codes over binary alphabet.

Alphabet reduction. A natural way of reducing the alphabet size of codes is via the well-known paradigm of *concatenated codes* [28]: A concatenated code is obtained by encoding the symbols of an "outer code" (using the coding method of the "inner code"). Specifically, let $\mathcal{C}_1 : \Sigma_1^{k_1} \to \Sigma_1^{n_1}$ be the outer code and $\mathcal{C}_2 : \Sigma_2^{k_2} \to \Sigma_2^{n_2}$ be the inner code, where $\Sigma_1 \equiv \Sigma_2^{k_2}$. Then, the concatenated code $\mathcal{C}' : \Sigma_2^{k_1 k_2} \to \Sigma_2^{n_1 n_2}$ is obtained by $\mathcal{C}'(x_1, ..., x_{k_1}) = (\mathcal{C}_2(y_1), ..., \mathcal{C}_2(y_{n_1}))$, where $x_i \in \Sigma_2^{k_2} \equiv \Sigma_1$ and $(y_1, ..., y_{n_1}) = \mathcal{C}_1(x_1, ..., x_{k_1})$. That is, first $\mathcal{C}_1$ is applied to the k_1-long sequence of k_2-symbol blocks, which are viewed as symbols of Σ_1, and then $\mathcal{C}_2$ is applied to the each of the resulting n_1 blocks (see Figure 2, where $k_1 = 4$, $n_1 = 6$, $k_2 = 8$ and $n_2 = 16$). 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.

[33] That is, it select uniformly at random $\overline{x}_1, \overline{x}_2, \overline{h}_1, \overline{h}_2 \in \mathcal{F}^m$ and $i_1, i_2 \in \mathcal{F}$ such that $\overline{x}_1 + i_1\overline{h}_1 = \overline{x}_2 + i_2\overline{h}_2$, and checks whether the value of the polynomial $L(\overline{x}_1, \overline{h}_1)$ at i_1 equals the value of the polynomial $L(\overline{x}_2, \overline{h}_2)$ at i_2.

[34] Indeed, it would have been more natural to present the code as a mapping from sequences over $\mathcal{F}$ to sequences over $\Sigma = \mathcal{F}^{d+1}$. Following the convention of using the same alphabet for both the information and the codeword, we just pack every $d+1$ elements of $\mathcal{F}$ as an element of Σ.

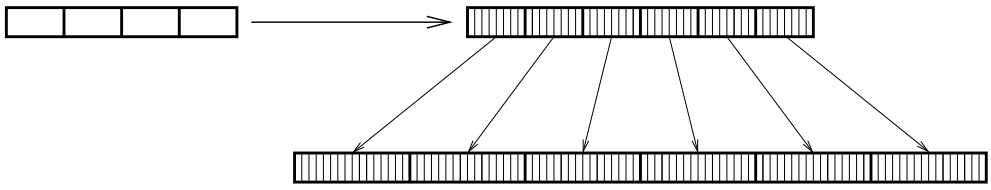

Figure 2: Concatenated codes. The outer (resp., inner) encoding is depicted by the horizontal arrow (resp., vertical arrows).

The problem, however, is that concatenated codes do not necessarily preserve local testability. Here, we shall use special features of the specific tester used for the outer codes. In particular, observe that, for each of the two queries made by the tester of $\mathcal{C} : \Sigma^k \to \Sigma^n$, the tester does not need the entire polynomial represented in $\Sigma = \mathcal{F}^{d+1}$, 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.[35]

In particular, for integers h, e such that $d + 1 = h^e$, Goldreich and Sudan used an encoding of the elements of $\Sigma = \mathcal{F}^{d+1} = \mathcal{F}^{h^e}$ by sequences of length $|\mathcal{F}|^{eh}$ over $\mathcal{F}$ (i.e., this inner code mapped h^e-long $\mathcal{F}$-sequences to $|\mathcal{F}|^{eh}$-long $\mathcal{F}$-sequences), and provided testing and recovery procedures (for this inner code) that make $O(e)$ queries [41, Sec. 3.3]. Note that the case of $e = 1$ and $|\mathcal{F}| = 2$ corresponds to the Hadamard code, and that a bigger constant e allows for shorter codes (e.g., for $|\mathcal{F}| = 2$, we have length $2^{eh} = 2^{e \cdot t^{1/e}}$, where t denotes the length of the encoded information). The resulting concatenated code, denoted $\mathcal{C}' : \mathcal{F}^{(d+1) \cdot k} \to \mathcal{F}^{n'}$, is a locally testable code over $\mathcal{F}$, and has length $n' = n \cdot O(d)^{eh} = n \cdot \exp((e \log d) \cdot d^{1/e})$. 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 $|\mathcal{F}| = \mathrm{poly}(\log k)$, which means that we have reduced the alphabet size considerably (from $|\mathcal{F}|^{d+1}$ to $|\mathcal{F}|$, where $d = \Theta(|\mathcal{F}|)$).

Finally, a *binary* locally testable code is obtained by concatenating $\mathcal{C}' : \mathcal{F}^{k'} \to \mathcal{F}^{n'}$ with the Hadamard code (which is used to encode elements of $\mathcal{F}$), while noting that the latter supports a "local recovery" property that suffices to emulate the tester for $\mathcal{C}'$. In particular, the tester of $\mathcal{C}'$ merely checks a linear (over $\mathcal{F}$) equation referring to a constant number of $\mathcal{F}$-elements, and for $\mathcal{F} = \mathrm{GF}(2^\ell)$, this can be emulated by checking *related* random linear combinations of the bits representing these elements, which in turn can be locally recovered (or rather self-corrected) from the Hadamard code. The final result is a locally testable (binary) code of nearly quadratic length; that is, the length is $n' \cdot 2^\ell = n' \cdot \mathrm{poly}(\log k)$, whereas the information contents is $k' \cdot \ell > k$ (and $n' \approx k^{2c/(c-1)} \cdot \exp(\widetilde{O}(\log k)^{1/2})$).[36] We comment that a version of this tester may use three queries, whereas 2-query locally testable *binary* codes are essentially impossible (cf., [13]).

3.2.2 Locally testable proofs of polynomial length: The PCP Theorem

The case of proofs is far more complex than that of codes: Achieving locally testable proofs of polynomial length is essentially the contents of the celebrated PCP Theorem of Arora, Lund,

[35] Indeed, this property is related to locally decodable codes (to be briefly discussed in Section 4.4). Here we need to recover one out of $|\mathcal{F}|$ specific linear combinations of the encoded $(d+1)$-long sequence of $\mathcal{F}$-symbols. In contrast, locally decodable refers to recovering one out of the $\mathcal{F}$-symbols of the original $(d+1)$-long sequence.

[36] Actually, the aforementioned result is only implicit in [41], since Goldreich and Sudan apply these ideas directly to a truncated version of the low-degree based code.

Motwani, Sudan and Szegedy [3], which asserts that *every set in $\mathcal{NP}$ has a PCP system of constant query complexity and logarithmic randomness complexity.*[37] The construction is analogous to (but far more complex than) the one presented in the case of codes:[38] First we construct locally testable proofs over a large alphabet, and next we compose such proofs with corresponding "inner" proofs (over a smaller alphabet, and finally over a binary one). Our exposition focuses on the construction of these proof systems and somewhat blurs the issues involved in their composition.

Teaching note: This subsection is significantly more complex than the rest of this section, and some readers may prefer to skip it and proceed directly to Section 3.3. Specifically, we proceed in four steps:

1. Introduce an NP-complete problem, denoted PVPP.
2. Present a PCP over large alphabet for PVPP.
3. Perform alphabet (and/or query complexity) reduction for PCPs.
4. Discuss the proof composition paradigm, which underlies the prior part.

(The presentation of Step 1-3 (which follows [64, Apdx. C] and [11]) is different from the standard presentation of [3].) The second and third steps are most imposing and complex, but the reader may benefit from the discussion of the proof composition paradigm (Step 4) even when skipping all prior steps. Our presentation of the composition paradigm follows [11], rather than the original presentation of [4, 3]. For further details regarding the proof composition paradigm, the reader is referred to [34, Sec. 9.3.2.2].

The partially vanishing polynomial problem (PVPP). As a preliminary step, we introduce the following NP-complete problem, for which we shall present a PCP. The input to the problem consists of a finite field $\mathcal{F}$, a subset $H \subset \mathcal{F}$ of size $|\mathcal{F}|^{1/15}$, an integer $m < |H|$, and a $(3m+4)$-variant polynomial $P : \mathcal{F}^{3m+4} \to \mathcal{F}$ of total degree $3m|H| + O(1)$. The problem is to determine whether there exists an m-variant ("assignment") polynomial $A : \mathcal{F}^m \to \mathcal{F}$ of total degree $m|H|$ such that $P'(x,y,z,\tau) \stackrel{\text{def}}{=} P(x,z,y,\tau,A(x),A(y),A(z))$ vanishes on $H^{3m} \times \{0,1\}^3$; that is,

$$P(x,z,y,\tau,A(x),A(y),A(z)) = 0 \text{ for every } x,y,z \in H^m \text{ and } \tau \in \{0,1\}^3 \subset H. \tag{5}$$

Note that the instance (i.e., the polynomial P) can be explicitly described by a sequence of $|\mathcal{F}|^{3m+4} \log_2 |\mathcal{F}|$ bits, whereas the solution sought can be explicitly described by a sequence of $|\mathcal{F}|^m \log_2 |\mathcal{F}|$ bits. We comment that the NP-completeness of the aforementioned problem can be proved via 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).[39] 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).

A PCP over large alphabet for PVPP. The proof that a given input P satisfies the condition in Eq. (5) consists of an m-variant polynomial $A : \mathcal{F}^m \to \mathcal{F}$ (which is supposed to be of total

[37] Recall that the proof complexity of PCPs is exponential in their randomness complexity (and linear in their query complexity).

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

[39] Specifically, $f(x,y,z,\sigma,\tau,\xi) = 1$ if and only if $x^\sigma \vee y^\tau \vee z^\xi$ appears as a clause in the given formula, where x^σ denotes x if $\sigma = 0$ and $\neg x$ otherwise.

degree $m|H|$) as well as $3m+1$ auxiliary polynomials $A_i : \mathcal{F}^{3m+1} \to \mathcal{F}$, for $i = 1, ..., 3m+1$ (each supposedly of degree $(3m|H| + O(1)) \cdot m|H|$). The polynomial A is supposed to satisfy Eq. (5); that is, $P(x, z, y, \tau, A(x), A(y), A(z)) = 0$ should hold for every $x, y, z \in H^m$ and $\tau \in \{0,1\}^3 \subset H$. Furthermore, $A_0(x, y, z, \tau) \stackrel{\text{def}}{=} P(x, z, y, \tau, A(x), A(y), A(z))$ should vanish on H^{3m+1} (i.e., $A_0(x, y, z, \tau) = 0$ for every $x, y, z \in H^m$ and $\tau \in H$). The auxiliary polynomials are given to assist the verification of the latter condition. In particular, A_i should vanish on $\mathcal{F}^i H^{3m+1-i}$, 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 $\mathcal{F}^{i-1}H^{3m+1-(i-1)}$, 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 a low-degree test. Recall that it can be performed by selecting a random line through $\mathcal{F}^m$, and testing whether A restricted to this line agrees with a degree $m|H|$ univariate polynomial).

2. Testing that, for $i = 1, ..., 3m+1$, the polynomial A_i is of total degree $d \stackrel{\text{def}}{=} (3m|H| + O(1)) \cdot m|H|$.

   (Here we select a random line through $\mathcal{F}^{3m+1}$, and test whether A_i restricted to this line agrees with a degree d univariate polynomial.)

3. Testing that, for $i = 1, ..., 3m+1$, the polynomial A_i agrees with A_{i-1} on $\mathcal{F}^{i-1}H\mathcal{F}^{3m+1-i}$, which implies that A_i agrees with A_{i-1} on $\mathcal{F}^{i-1}H^{3m+1-(i-1)}$.

   This is done by uniformly selecting $r' = (r_1, ..., r_{i-1}) \in \mathcal{F}^{i-1}$ and $r'' = (r_{i+1}, ..., r_{3m+1}) \in \mathcal{F}^{3m+1-i}$, 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 univariate polynomial of degree d.[40]

   We stress that the values of A_0 are computed according to the given polynomial P by accessing A at the appropriate 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 $\mathcal{F}^{3m+1}$.

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

The foregoing tester may be viewed as making $O(m|\mathcal{F}|)$ queries to an oracle of length $|\mathcal{F}|^m + (3m+1) \cdot |\mathcal{F}|^{3m+1}$ over the alphabet $\mathcal{F}$, or alternatively, as making $O(m|\mathcal{F}| \log |\mathcal{F}|)$ binary queries to a binary oracle of length $O(m \cdot |\mathcal{F}|^{3m+1} \log |\mathcal{F}|)$. We mention that the foregoing description (which follows [64, Apdx. C]) is somewhat different than the original presentation in [3], which in turn follows [5, 6, 27].[41]

Note that we have already obtained a highly non-trivial tester. It makes $O(m|\mathcal{F}| \log |\mathcal{F}|)$ queries to a proof of length $\widetilde{O}(m \cdot |\mathcal{F}|^{3m+1})$ in order to verify a claim regarding an input of length $n \stackrel{\text{def}}{=} |\mathcal{F}|^{3m+4} \log_2 |\mathcal{F}|$. Using $m = \Theta(\log n / \log \log n)$, $|H| = \log n$ and $|\mathcal{F}| = \text{poly}(\log n)$, which satisfies

[40] Thus, effectively, we are self-correcting the values at H (on the said line), based on the values at $\mathcal{F}$ (on that line).

[41] The point is that the sum-check, which originates in [53], is replaced here by an analogous process (which is non-sequential in nature).

$m < |H| = |\mathcal{F}|^{1/15}$, we have obtained a *tester of poly-logarithmic query complexity and polynomial proof complexity* (equivalently, logarithmic randomness complexity).[42]

Although the foregoing tester is highly non-trivial, it falls short from our aim, because it employs a *non-constant number of queries to a proof-oracle over a non-constant alphabet*. Of course, we can convert the latter alphabet to a binary alphabet by increasing the number of queries, but actually the original proof of the PCP Theorem went in the opposite direction and reduce the number of queries by "packing" them into a constant number of queries to an oracle over an even larger alphabet (see the "parallelization technique" below). Either way, we are faced with the problem of *reducing the total amount of information obtained from the oracle*.

Alphabet (and/or query complexity) reduction for PCPs. To further reduce the query complexity, we invoke the "proof composition" paradigm, introduced by Arora and Safra [4] (and further discussed at the end of the current subsection). Specifically, we compose an "outer" tester (e.g., the foregoing tester) with an "inner" tester that locally checks the residual condition that the "outer" would have checked (regarding the answers it would have obtained). That is, rather than letting the "outer" verifier read (small) portions of the proof-oracle and decide accordingly, we let the "inner" verifier probe these portions and check whether the "outer" verifier would have accepted based on them. This composition is not straightforward, 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, since it is not clear how the "inner" tester can operate without reading its entire input. The problem can be resolved by using a "proximity tester" (i.e., a PCP of Proximity)[43] as an "inner" tester, provided that it suffices to have such a proximity test (for the answers to the "outer" tester). Thus, the challenge is to reach a situation in which the "outer" tester is "robust" in the sense that, when the assertion is false, the answers obtained by this tester are far from being convincing (i.e., far from any sequence of answers that is accepted by this tester). Two approaches towards obtaining such robust testers are known.

- One approach, introduced in [3], 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 in an error correcting format. Thus, robustness is guaranteed by the fact that the answers are presented as a sequence consisting of a constant number of codewords, and so any two (properly formatted) sequences are at constant relative distance of one another.

  The implementation of this approach consists of two steps. The first step is to convert the "outer" tester that makes $t = \mathrm{poly}(\log \ell)$ queries to an oracle $\pi : [\ell] \to \{0,1\}$ into a tester that makes a constant number of queries to an oracle that maps $[\mathrm{poly}(\ell)]$ to $\{0,1\}^{\mathrm{poly}(t)}$. This step uses the so-called parallelization technique, which replaces each possible t-sequence of queries by a (low degree) curve that passes through these t queries as well as through a random point (cf. [52, 3]). The new proof-oracle answers each such curve C with a (low degree) univariate polynomial p_C that is supposed to describe the values of (a low degree extension π' of) π at all $\mathrm{poly}(t)$ points that reside on C (i.e., $p_C(i) = \pi'(C(i))$. The consistency of these p_C's with π is check by selecting a random curve C, and comparing the value that p_C assigns a random point on C to the value assigned to this point by π' (i.e., the low-degree extension of π).[44]

[42] In fact, the proof complexity is sub-linear, since $\widetilde{O}(m \cdot |\mathcal{F}|^{3m+1}) = o(n)$.

[43] See Section 2.3.3.

[44] **Advanced comment:** Specifically, we associate $[\ell]$ with H^m, where $m \approx |H|$ and H resides in a finite field $\mathcal{F}$

In the second step, an error correcting code is applied to the poly(t)-bit long answers provided by the foregoing oracle, while assuming that the "inner (proximity) verifier" can handle inputs that are presented in this format (i.e., that it can test an input that is presented in a constant number of parts, where each part is encoded separately).[45]

- An alternative approach, pursued and advocated in [11], is to take advantage of the specific structure of the queries, "bundle" the answers together (into a constant number of bundles) and 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. Furthermore, the robustness of individual bundles is inherited by any constant sequence of bundles, and so there is no need to use error correcting codes (on top of the bundled answers).

Hence, while the first approach relies on a general technique of parallelization (and, historically (see Footnote 45), also on the specifics of the inner verifier), the second approach refers explicitly to the notion of robustness and relies on the specifics of the outer verifier. An advantage of the second approach is that it almost preserves the length of the proofs (whereas the first approach may square this length). We will outline the second approach next, but warn that this terse description may be hard to follow.

First, we show how the queries of the foregoing *tester for PVPP* can be "bundled" such that the $O(m)$ sub-tests of this tester can be performed by inspecting a constant number of bundles. In particular, we consider the following "bundling" that accommodates the $3m+1$ different sub-tests performed in Step (3): Consider $B : \mathcal{F}^{3m+1} \to \mathcal{F}^{3m+1}$ such that

$$\begin{aligned} &B(x_1, ..., x_{3m+1}) \\ &\stackrel{\text{def}}{=} (A_1(x_1, x_2, ..., x_{3m+1}), A_2(x_2, ..., x_{3m+1}, x_1), ..., A_{3m+1}(x_{3m+1}, x_1, ..., x_{3m})) \end{aligned} \tag{6}$$

such that $|\mathcal{F}| = \text{poly}(t, |H|)$. (We stress that m, H and $\mathcal{F}$ used here are different from those used in the foregoing description of the PCP for PVPP.) For every sequence of queries $\overline{q} = (q_1, ..., q_t) \in (H^m)^t$ made by the original verifier and every $r \in \mathcal{F}^m$, we consider the degree $t+1$ curve $C_{\overline{q},r} : \mathcal{F} \to \mathcal{F}^m$ such that $C_{\overline{q},r}(0) = r$ and $C_{\overline{q},r}(i) = q_i$ for every $i \in [t] \subset \mathcal{F}$. Hence, the set of curves corresponds to $\Omega \times \mathcal{F}^m$, where Ω is the set of all possible outcomes of the internal coin tosses of the original verifier. The new proof-oracle consists of a function $\pi' : \mathcal{F}^m \to \mathcal{F}$, which is supposed to be a degree $m|H|$ extension of the original proof π, viewed as a Boolean function $\pi : H^m \to \{0,1\}$, as well as univariate polynomials of degree $m|H| \cdot (t+1)$ that are supposed to represent the restrictions of π' to all $|\Omega \times \mathcal{F}^m|$ curves (i.e., the polynomial $p_C : \mathcal{F} \to \mathcal{F}$ that corresponds to the curve C is supposed to satisfy $p_C(i) = \pi'(C(i))$ for every $i \in \mathcal{F}$). The new verifier will

1. test that π' has degree $m|H|$;
2. test that π' matches the univariate polynomials by selecting a random point $i \in \mathcal{F}$ on a random curve C and comparing the value given by the corresponding univariate polynomial p_C to the value given by π' (i.e., cheching that $p_C(i) = \pi'(C(i))$ holds); and
3. select a random curve $C = C_{\overline{q},r}$ and emulate the original tester based on the values $p_C(1), ..., p_C(t)$ obtained from the polynomial that corresponds to this curve.

Due to the randomization of the curves via their value at zero, it holds that a random point on a random curve is distributed almost uniformly in $\mathcal{F}^m$, where the possible slackness is due to the first t points on the curve. The analysis is based on the fact that if π' has degree $m|H|$ and the polynomial that corresponds to a curve does not agree with it at some point, then they disagree on most of the points.

[45] The aforementioned assumption holds trivially in case one uses a general-purpose "proximity tester" (e.g., a PCP of Proximity (a.k.a. an Assignment Tester) for sets in $\mathcal{P}$) as done in [24]. But the aforementioned approach can be applied (and, in fact, was originally applied) using a specific "proximity tester" that can only handle inputs presented in one specific format (cf. [3]).

and perform all $3m+1$ tests of Step (3) by selecting uniformly $(r_2, ..., r_{3m+1}) \in \mathcal{F}^{3m}$ and querying B at $(e, r_2, ..., r_{3m+1})$ and $(r_{3m+1}, e, ..., r_{3m})$ for all $e \in H$. Thus, all $3m+1$ tests of Step (3) can be performed by retrieving the $2 \cdot |\mathcal{F}|$ values of B on two *axis parallel* random line through $\mathcal{F}^{3m+1}$ (i.e., the lines $(\cdot, r_2, ..., r_{3m+1})$ and $(r_{3m+1}, \cdot, r_2, ..., r_{3m})$).[46] Likewise, all $3m+1$ tests of Step (2) can be performed by retrieving the $|\mathcal{F}|$ values of B on a single (arbitrary) random line through $\mathcal{F}^{3m+1}$. (The test of Step (1), which refers to A, remains intact, whereas the test of Step (4) is conducted on B rather than on A_{3m+1}.) Lastly, observe that these 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 the $|\mathcal{F}|$-long sequence of values of A_i on an appropriate random line will be far from satisfying the corresponding predicate. This robustness feature is inherited by B, since each symbol of B encodes the corresponding values of all A_i's. Hence, we have bundled $O(m)$ tests that refer to $O(m)$ different functions (i.e., the A_i's and A) into four tests that refer to two functions (i.e., B and A), where each of these tests queries one (or both) of the functions for its value at $O(|\mathcal{F}|)$ points.[47]

Next, we encode the symbols of B (resp., of A) by a good binary error-correcting, and obtain a binary function B' (resp., A') that preserves the robustness up to a constant factor (which equals the relative distance of the code). Specifically, we may replace $A : \mathcal{F}^m \to \mathcal{F}$ and $B : \mathcal{F}^{3m+1} \to \mathcal{F}^{3m+1}$ by $A' : \mathcal{F}^m \times [O(\log |\mathcal{F}|)] \to \{0,1\}$ and $B' : \mathcal{F}^{3m+1} \times [O(\log |\mathcal{F}|^{3m+1})] \to \{0,1\}$, and conduct all all tests by making $O(m^2 |\mathcal{F}| \log |\mathcal{F}|)$ queries to A' and B' (since each query to $A : \mathcal{F}^m \to \mathcal{F}$ (resp., to $B : \mathcal{F}^{3m+1} \to \mathcal{F}^{3m+1}$) is replace by $O(\log |\mathcal{F}|)$ queries to A' (resp., $O(m \log |\mathcal{F}|)$ queries to B')). The resulting *robustness feature* asserts that if the original polynomial P had no solution (i.e., an A satisfying Eq. (5)), then the answers obtained by the tester will be far from satisfying the residual decision predicate of the tester.

Now, if the robustness feature of the resulting ("outer") tester fits the proximity testing feature of the "inner tester" (i.e., the threshold determining what is "far" w.r.t robustness is greater than or equal to the threshold of "far" w.r.t proximity), then 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 (in n) to polynomial in a double-logarithm function (in n). At this point we can afford the Hadamard-Based proof tester (because the overhead in the proof length will only be exponential in $\text{poly}(\log \log n) = O(\log n)$), and obtain a locally testable proof of polynomial (in n) length. That is, we compose the $\text{poly}(\log \log)$-query tester (acting as an outer tester) with the Hadamard-Based tester (acting as an inner tester), and obtain a locally testable proof of polynomial length (as asserted by the PCP Theorem).

On the proof composition paradigm. The PCP Theorem asserts a PCP system for $\mathcal{NP}$ that simultaneously achieve the minimal possible randomness and query complexity (up to a multiplica-

[46] Indeed, the values of $B(e, r_2, ..., r_{3m+1})$ and $B(r_{3m+1}, e, r_2, ..., r_{3m})$ yield the values of $A_{i-1}(r_i, ..., r_{3m+1}, e, r_2, ..., r_{i-1})$ and $A_i(r_i, ..., r_{3m+1}, e, r_2, ..., r_{i-1})$ for every $i \in [3m+1]$. Recall, however, that the values of A_0 are determined based on A. Hence, for emulating the first of these tests (i.e., the test corresponding to $i = 1$), we use both B and A.

[47] Actually, the fourth test (corresponding to Step (4)) queries B at a single point. Recall that Step (1) queries A on a random line, Step (2) queries B on a random line, and Step (3) queries B (and A) on two random axis-parallel lines.

tive factor).[48] The foregoing construction obtains this remarkable result by combining two different PCPs: the first PCP obtains logarithmic randomness but uses poly-logarithmically many queries, whereas the second PCP uses a constant number of queries but has polynomial randomness complexity. We stress that *each of these two PCP systems is highly non-trivial and very interesting by itself*. We also highlight the fact that these PCPs are combined using a very simple composition method (which refers to auxiliary properties such as robustness and proximity testing). Details follow.[49]

Loosely speaking, the proof composition paradigm refers to composing two proof systems such that the "inner" verifier is used for probabilistically verifying the acceptance criteria of the "outer" verifier. That is, the combined verifier selects coins for the "outer" verifier, determines the corresponding locations that the "outer" verifier would have inspected (in the proof), and verifies that the "outer" verifier would have accepted the values that reside in these locations. The latter verification is performed by invoking the "inner" verifier, *without reading the values residing in all the aforementioned locations*. Indeed, the aim is to conduct this ("composed") verification while using much fewer queries than the query complexity of the "outer" proof system. In particular, the inner verifier cannot afford to read its input, which makes the composition more subtle than the term suggests.

In order for the proof composition to work, the verifiers being combined should satisfy some auxiliary conditions. Specifically, the *outer* verifier should be robust in the sense that its soundness condition guarantee that, with high probability, the oracle answers are "far" from satisfying the residual decision predicate (rather than merely not satisfying it).[50] The *inner* verifier is given oracle access to its input and is charged for each query made to it, but is only required to reject (with high probability) inputs that are far from being valid (and, as usual, accept inputs that are valid). That is, the inner verifier is actually a verifier of proximity (i.e., a PCP of Proximity, as defined in Section 2.3.3).

Composing two such PCPs yields a new PCP, where the new proof-oracle consists of the proof-oracle of the "outer" system and a sequence of proof-oracles for the "inner" system (one "inner" proof per each possible random-tape of the "outer" verifier). The resulting verifier selects coins for the outer-verifier and uses the corresponding "inner" proof in order to verify that the outer-verifier would have accepted under this choice of coins. Note that such a choice of coins determines locations in the "outer" proof that the outer-verifier would have inspected, and the combined verifier provides the inner-verifier with oracle access to these locations (which the inner-verifier considers as its input) as well as with oracle access to the corresponding "inner" proof (which the inner-verifier considers as its proof-oracle).

The quantitative effect of such a composition is easy to analyze. Specifically, composing an outer-verifier of randomness-complexity r' and query-complexity q' with an inner-verifier of randomness-complexity r'' and query-complexity q'' yields a PCP of randomness-complexity $r(n) = r'(n) + r''(q'(n))$ and query-complexity $q(n) = q''(q'(n))$, because $q'(n)$ represents the length of the input (oracle) that is accessed by the inner-verifier. Thus, assuming $q''(m) \ll m$, the query com-

[48] The claim of minimality assumes that $\mathcal{P} \neq \mathcal{NP}$, and for the claim regarding randomness complexity refers to low query complexity. The point is that a PCP system of randomness complexity $r(n) = O(\log n)$ and query complexity $q(n)$ yields an NP-proof system that utilizes proofs of length $2^{r(n)} \cdot q(n)$.

[49] Our presentation of the composition paradigm follows [11], rather than the original presentation of [4, 3]. A more detailed overview of the composition paradigm is available in [34, Sec. 9.3.2.2].

[50] Furthermore, the latter predicate, which is well-defined by the non-adaptive nature of the outer verifier, must have a circuit of size that is at most polynomial in the number of queries.

plexity is significantly decreased (from $q'(n)$ to $q''(q'(n))$), while the increase in the randomness complexity is moderate provided that $r''(q'(n)) \ll r'(n)$. Furthermore, the verifier resulting from the composition inherits the robustness features of the inner verifier, which is important in case we wish to compose the resulting verifier with another inner-verifier.

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. Currently, this taxonomy is relevant mainly for second-level discussions and review of some past works.[51]

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

T1-nearly linear: A very liberal notion, which seems at the verge of an abuse of the term, refers to a sequence of functions $f_\epsilon : \mathbb{N} \to \mathbb{N}$ 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 [59] and the simpler locally testable code of Goldreich and Sudan [41, 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 $\epsilon : \mathbb{N} \to [0,1]$ that tends 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 constant $c \in (0,1)$.
   The locally testable codes and proofs of [41, 18, 11] have nearly linear length in this sense. Specifically, in [41, Sec. 4-5] and [18] any $c > 1/2$ will do, whereas in [11] any $c > 0$ will do.
3. $\epsilon(n) = \frac{\exp((\log\log n)^c)}{\log n}$ for some constant $c \in (0,1)$.
   Note that $\mathrm{poly}(\log\log n) < \exp((\log\log n)^c) < (\log n)^{o(1)}$, for any constant $c \in (0,1)$.
4. $\epsilon(n) = \frac{\mathrm{poly}(\log\log n)}{\log n}$, which corresponds to $f(n) = q(\log n)\cdot n$, where $q(m) = \exp(\mathrm{poly}(\log m))$.
   Here near-linearity means as linearity up to a quasi-poly-logarithmic factor, and one is tempted to view it as a relaxation of the following type (T3).

Indeed, the case in which $\epsilon(n) = \frac{O(\log\log n)}{\log n}$ deserves a special category, presented next.

T3-nearly linear: The strongest notion interprets near-linearity as linearity up to a poly-logarithmic factor; that is, $f(n) = \widetilde{O}(n) \stackrel{\text{def}}{=} \mathrm{poly}(\log n)\cdot n$, which corresponds to the case of $f(n) \leq n^{1+\epsilon(n)}$ with $\epsilon(n) = O(\log\log n)/\log n$.

The results of [17, 22, 65, 66] refer to this notion.

[51] Things were different when the original version of this text [33] was written. At that time, only T2-nearly linear length was know for $O(1)$-local testability, and the T3-nearly linear result achieved later by Dinur [22] seemed a daring conjecture (which was, nevertheless, stated in [33, Conj. 3.3]).

We note that while [17, 22, 65, 66] achieve T3-nearly linear length, the low-error results of [57, 23] only achieve T2-nearly linear length.

3.3.1 Local testability with nearly linear length

The celebrated gap amplification technique of Dinur [22] is best known for providing an alternative proof of the PCP Theorem (which asserts that *every set in $\mathcal{NP}$ has a PCP system of constant query complexity and logarithmic randomness complexity*). However, applying this technique to a PCP that was (previously) provided by Ben-Sasson and Sudan [17] yields locally testable codes and proofs of T3-nearly linear length. In particular, the overhead in the code and proof length is only polylogarithmic in the length of the primal object (which establishes [33, Conj. 3.3]).

Theorem 11 (Dinur [22], building on [17]): *There exists a constant q and a poly-logarithmic function $f : \mathbb{N} \to \mathbb{N}$ such that there exist q-locally testable codes and proofs* (for `SAT`) *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 PCP system asserted in Theorem 11 is obtained by applying the gap amplification method of Dinur [22] (reviewed in Section 3.3.2) to the PCP system of Ben-Sasson and Sudan [17]. We mention that the PCP system (for NP) of Ben-Sasson and Sudan [17] is based on the NP-completeness of a certain code (of length $n = \widetilde{O}(k)$), and on a randomized reduction of testing whether a given n-bit long string is a codeword to a constant number of similar tests that refer to $\sqrt{n}$-bit long strings. Applying this reduction $\log \log n$ times yields a PCP of query complexity $\mathrm{poly}(\log n)$ and length $\widetilde{O}(n)$, which in turn yields a 3-query "weak PCP with soundness error $1 - 1/\mathrm{poly}(\log n)$".

The PCP system of Theorem 11 can be adapted to yield a PCP of Proximity with the same parameters, which (as shown in Section 2.3.3) yields a (weak) locally testable code of similar parameters (i.e., constant number of queries and length $n = \widetilde{O}(k)$). Recall that this transformation of PCP of Proximity to locally testable codes only works for the weak version of the latter notion. A strong locally testable code of similar parameters was only obtained later (by Viderman [65, 66]).

Is a polylogarithmic overhead the best one can get? In the original version of this survey [33], we conjectured that a polylogarithmic (length) overhead is inherent to local testability (or, at least, that linear length $O(1)$-local testability is impossible). We currently have mixed feelings with respect to this conjecture (even when confined to proofs), and thus rephrase it as an open problem.

Open Problem 12 (local testability in linear length): *Determine whether there exist locally testable codes and proofs of linear length.*

3.3.2 The gap amplification method

Essentially, Theorem 11 is proved by applying the gap amplification method (of Dinur [22]) to the (weak) PCP system constructed by Ben-Sasson and Sudan [17]. The latter PCP system has length $\ell(k) = \widetilde{O}(k)$, but its soundness error is $1 - 1/\mathrm{poly}(\log k)$ (i.e., its rejection probability is at least $1/\mathrm{poly}(\log k)$). Each application of the gap amplification step *doubles the rejection probability while essentially maintaining the initial complexities*. That is, in each step, the constant query complexity of the verifier is preserved and its randomness complexity is increased only by a constant term (and

so the length of the PCP oracle is increased only by a constant factor). Thus, starting from the system of [17] and applying $O(\log \log k)$ amplification steps, we essentially obtain Theorem 11. (Note that a PCP system of polynomial length can be obtained by starting from a trivial "PCP" system that has rejection probability $1/\mathrm{poly}(k)$, and applying $O(\log k)$ amplification steps.)[52]

In order to rigorously describe the aforementioned process we need to *redefine PCP systems so as to allow arbitrary soundness error*. In fact, for technical reasons, it is more convenient to describe the process in terms of an iterated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labeled) graphs such that the vertices correspond to (non-Boolean) variables and the edges are associated with constraints.

Definition 13 (CSP with 2-variable constraints): *For a fixed finite set Σ, an* instance of CSP *consists of a graph $G = (V, E)$, which may have parallel edges and self-loops, and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e \in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0,1\}$. The* value of an assignment *$\alpha : V \to \Sigma$ is the number of constraints satisfied by α; that is, the value of α is $|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 1\}|$. We denote by $\mathtt{vlt}(G, \Phi)$* (standing for violation) *the fraction of* unsatisfied *constraints under the best possible assignment; that is,*

$$\mathtt{vlt}(G, \Phi) = \min_{\alpha : V \to \Sigma} \left\{ \frac{|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 0\}|}{|E|} \right\}. \tag{7}$$

For various functions $\tau : \mathbb{N} \to (0,1]$, we will consider the promise problem $\mathtt{gapCSP}^{\Sigma}_{\tau}$, having instances as above, such that the YES*-instances are fully satisfiable instances* (i.e., $\mathtt{vlt} = 0$) *and the* NO*-instances are pairs (G, Φ) for which $\mathtt{vlt}(G, \Phi) \geq \tau(|G|)$ holds, where $|G|$ denotes the number of edges in G.*

Note that 3SAT is reducible to $\mathtt{gapCSP}^{\Sigma_0}_{\tau_0}$ for $\Sigma_0 = \{\mathtt{F}, \mathtt{T}\}^3$ and $\tau_0(m) = 1/m$ (e.g., replace each clause of the 3SAT instance by a vertex, and use edge constraints that enforce mutually consistent and satisfying assignments to each pair of clauses).[53] Furthermore, the PCP system of [17] yields a reduction of 3SAT to $\mathtt{gapCSP}^{\Sigma_0}_{\tau_1}$ for $\tau_1(m) = 1/\mathrm{poly}(\log m)$ where the size of the graph is nearly linear in the length of the input formula.

Our goal is to reduce $\mathtt{gapCSP}^{\Sigma_0}_{\tau_0}$ (or rather $\mathtt{gapCSP}^{\Sigma_0}_{\tau_1}$) to $\mathtt{gapCSP}^{\Sigma}_{c}$, for some fixed finite Σ and constant $c > 0$, where in the case of $\mathtt{gapCSP}^{\Sigma_0}_{\tau_1}$ we wish the reduction to preserve the length of the instance up to a polylogarithmic factor.[54] The PCP Theorem (resp., a PCP of nearly linear length) follows by showing a simple PCP system for $\mathtt{gapCSP}^{\Sigma}_{c}$ (e.g., the PCP verifier selects a random edge and checks whether the pair of values assigned to its endpoints by the alleged proof satisfies the constraint associated with this edge).[55] As noted before, the reduction is obtained by repeated applications of an amplification step that is captured by the following lemma.

[52] See Exercise 7.

[53] Given the instance $\wedge_{i \in [m]} \psi_i$, we construct a graph $G = ([m], E)$ such that vertices i and j are connected by an edge if and only if ψ_i and ψ_j have some common variable. In this case the constraint $\phi_{(i,j)} : \Sigma_0^2 \to \{0,1\}$ is such that $\phi_{(i,j)}(\sigma, \tau) = 1$ if and only if $\psi_i(\sigma) = \psi_j(\tau)$ and the values assigned to the common variable are identical. For example, if $\psi_i = x \vee y \vee z$ and $\psi_j = u \vee \neg x \vee \neg v$, then $\phi_{(i,j)}(\sigma, \tau) = 1$ if and only if $\sigma_1 \vee \sigma_2 \vee \sigma_3$ and $\tau_1 \vee \neg\tau_2 \vee \neg\tau_3$ and $\sigma_1 = \tau_2$.

[54] Hence, for some fixed Σ and constant $c > 0$, the problem $\mathtt{gapCSP}^{\Sigma}_{c}$ is NP-complete. As shown in Exercise 8, this cannot be the case if $|\Sigma| = 2$, unless $\mathcal{P} = \mathcal{NP}$.

[55] For $\Sigma \equiv \{0,1\}^{\ell}$, given a $\mathtt{gapCSP}^{\Sigma}_{c}$ instance (G, Φ), consider the PCP oracle $\pi : [n] \times [\ell] \to \{0,1\}$, where n denotes the number of vertices in G. The verifier selects a random edge (u,v) in G, obtains $\sigma = \pi(u,1) \cdots \pi(u,\ell)$ and $\tau = \pi(v,1) \cdots \pi(v,\ell)$, and checks whether $\phi_{(u,v)}(\sigma, tau) = 1$.

Lemma 14 (amplifying reduction of gapCSP to itself): *For some finite Σ and constant $c > 0$, there exists a polynomial-time computable function f such that, for every instance (G, Φ) of $\mathtt{gapCSP}^{\Sigma}$, it holds that $(G', \Phi') = f(G, \Phi)$ is an instance of $\mathtt{gapCSP}^{\Sigma}$ and the two instances are related as follows:*

1. *If $\mathtt{vlt}(G, \Phi) = 0$, then $\mathtt{vlt}(G', \Phi') = 0$.*
2. *$\mathtt{vlt}(G', \Phi') \geq \min(2 \cdot \mathtt{vlt}(G, \Phi), c)$.*
3. *$|G'| = O(|G|)$.*

That is, satisfiable instances are mapped to satisfiable instances, whereas instances that violate a ν fraction of the constraints are mapped to instances that violate at least a $\min(2\nu, c)$ fraction of the constraints. Furthermore, the mapping increases the number of edges (in the instance) by at most a constant factor. We stress that both Φ and Φ' consists of Boolean constraints defined over Σ^2. Thus, by iteratively applying Lemma 14 for a logarithmic (resp., double-logarithmic) number of times, we reduce $\mathtt{gapCSP}^{\Sigma}_{\tau_0}$ (resp., $\mathtt{gapCSP}^{\Sigma}_{\tau_1}$) to $\mathtt{gapCSP}^{\Sigma}_{c}$.

Teaching note: The rest of this subsection is also quite complex, and some readers may prefer to skip it and proceed directly to Section 4.

Outline of the proof of Lemma 14: Before turning to the proof, let us highlight the difficulty that it needs to address. Specifically, the lemma asserts a "violation amplifying effect" (i.e., Items 1 and 2), while maintaining the alphabet Σ and allowing only a moderate increase in the size of the graph (i.e., Item 3). Waiving the latter requirements allows a relatively simple proof that mimics (an augmented version of) the "parallel repetition" of the corresponding PCP. Thus, the challenge is significantly decreasing the "size blow-up" that arises from parallel repetition and maintaining a fixed alphabet. The first goal (i.e., Item 3) calls for a suitable derandomization, and indeed we shall use a "pseudorandom" generator based on random walks on expander graphs. The second goal (i.e., fixed alphabet) can be handled by using the proof composition paradigm, which was outlined in Section 3.2.2.

The lemma is proved by presenting a three-step reduction. The first step is a pre-processing step that makes the underlying graph suitable for further analysis (e.g., the resulting graph will be an expander). The value of vlt may decrease during this step by a constant factor. The heart of the reduction is the second step in which we can increase vlt by any desired constant factor. This is done by a construction that corresponds to taking a random walk of constant length on the current graph. The latter step also increases the alphabet Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems; e.g., the one presented in Section 3.1.2). Details follow.

We first stress that the aforementioned Σ and c, as well as the auxiliary parameters d and t (to be introduced in the following two paragraphs), are fixed constants that will be determined such that various conditions (which arise in the course of our argument) are satisfied. Specifically, t will be the last parameter to be determined (and it will be made greater than a constant that is determined by all the other parameters).

We start with the pre-processing step. Our aim in this step is to reduce the input (G, Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1, Φ_1) such that G_1 is a d-regular expander graph.[56] Furthermore,

[56] A *d-regular graph* is a graph in which each vertex is incident to exactly d edges. Loosely speaking, an expander

each vertex in G_1 will have at least $d/2$ self-loops, the number of edges will be preserved up to a constant factor (i.e., $|G_1| = O(|G|)$), and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. This step is quite simple: essentially, the original vertices are replaced by expanders of size proportional to their degree, and a big (dummy) expander is "superimposed" on the resulting graph. (The constraints associated with the edges of the former expanders mandate equality, whereas the the constraints associated with the edges of the latter expander are trivial (i.e., require nothing).)

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. The intuition underlying this step is that the probability that a random (t-edge long) walk on the expander G_1 intersects a fixed set of edges is closely related to the probability that a random sample of (t) edges intersects this set. Thus, we may expect such walks to hit a violated edge with probability that is at least $\min(\Theta(t \cdot \nu), c)$, where ν is the fraction of violated edges. Indeed, the current step consists of reducing the instance (G_1, Φ_1) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\mathtt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$ and the following holds:

1. The vertex set of G_2 is identical to the vertex set of G_1, and each t-edge long path in G_1 is replaced by a corresponding edge in G_2, which is thus a d^t-regular graph.

2. The constraints in Φ_2 view each element of Σ' as a Σ-labeling of the ("distance $\leq t$") neighborhood of a vertex, and mandates that the two corresponding labelings (of the endpoints of the G_2-edge) are consistent as well as satisfy Φ_1. That is, the following two types of conditions are enforced by the constraints of Φ_2:

   (consistency): If vertices u and w are connected in G_1 by a path of length at most t and vertex v resides on this path, then the Φ_2-constraint associated with the G_2-edge between u and w mandates the equality of the entries corresponding to vertex v in the Σ'-labeling of vertices u and w.

   (satisfying Φ_1): If the G_1-edge (v, v') is on a path of length at most t starting at u, then the Φ_2-constraint associated with the G_2-edge that corresponds to this path enforces the Φ_1-constraint that is associated with (v, v').

Clearly, $|G_2| = d^{t-1} \cdot |G_1| = O(|G_1|)$, because d is a constant and t will be set to a constant. (Indeed, the relatively moderate increase in the size of the graph corresponds to the low randomness-complexity of selecting a random walk of length t in G_1.)

Turning to the analysis of this step, we note that $\mathtt{vlt}(G_1, \Phi_1) = 0$ implies $\mathtt{vlt}(G_2, \Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\mathtt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1)), c)$. Here we merely provide a rough intuition and refer the interested reader to [22]. We may focus on any Σ'-labeling of the vertices of G_2 that is consistent with some Σ-labeling of G_1, because relatively few inconsistencies (among the Σ-values assigned to a vertex by the Σ'-labeling of other vertices) can be ignored, while relatively many such inconsistencies yield violation of the "equality constraints" of many edges in G_2. Intuitively, relying on the hypothesis that G_1 is an expander, it follows that the set of violated edge-constraints (of Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated (because each edge-constraint of Φ_1 is enforced by many edge-constraints of Φ_2). The

graph has the property that each cut (i.e., partition of its vertex set) has relatively many edges crossing it. An equivalent definition, also used in the actual analysis, is that all the eigenvalues of the corresponding adjacency matrix, except for the largest one (which equals d), have absolute value that is bounded away from d.

point is that *any set F of edges of G_1 is likely to appear on a* $\min(\Omega(t) \cdot |F|/|G_1|, \Omega(1))$ *fraction of the edges of* G_2 (i.e., t-paths of G_1). (Note that the claim would have been obvious if G_1 were a complete graph, but it also holds for an expander.)[57]

The factor of $\Omega(\sqrt{t})$ gained in the second step makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step). Furthermore, for a suitable choice of the constant t, the aforementioned gain yields an overall constant factor amplification (of $\mathtt{vlt}$). However, so far we obtained an instance of $\mathtt{gapCSP}^{\Sigma'}$ rather than an instance of $\mathtt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\mathtt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\mathtt{gapCSP}^{\Sigma'}$ as a PCP-system,[58] and composing it with an inner-verifier using the proof composition paradigm outlined in Section 3.2.2. We stress that the inner-verifier used here needs only handle instances of constant size (i.e., having description length $O(d^t \log |\Sigma|)$), and so the verifier presented in Section 3.1.2 will do. The resulting PCP-system uses randomness $r \stackrel{\text{def}}{=} \log_2 |G_2| + O(d^t \log |\Sigma|)^2$ and a constant number of binary queries, and has rejection probability $\Omega(\mathtt{vlt}(G_2, \Phi_2))$, *which is independent of the choice of the constant* t. Moving back to the world of $\mathtt{gapCSP}$, for $\Sigma = \{0,1\}^{O(1)}$, we can obtain an instance of $\mathtt{gapCSP}^{\Sigma}$ that has a $\Omega(\mathtt{vlt}(G_2, \Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance (which is used as the output (G', Φ') of the three-step reduction) is $O(2^r) = O(|G_2|)$, where the equality uses the fact that d and t are constants. Recalling that $\mathtt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1)), c)$ and $\mathtt{vlt}(G_1, \Phi_1) = \Omega(\mathtt{vlt}(G, \Phi))$, this completes the (outline of the) proof of the entire lemma. ■

Reflection. In contrast to the proof outlined in Section 3.2.2, which combines two remarkable constructs by using a simple composition method, the current proof of the PCP Theorem is based on developing a powerful amplification method that improves the quality of the system to which it is applied. This new method, captured by the amplification step (Lemma 14), does not obtain the best aspects of the given systems, but rather obtains a better system than the one given. However, the quality-amplification offered by Lemma 14 is rather moderate, and thus many applications are required in order to derive the desired result. Taking the opposite perspective, one may say that remarkable results are obtained by a gradual process of many moderate amplification steps.

4 Chapter notes

The term "locally testable proof" was introduced in [33] with the intension of matching the term "locally testable codes". As started at the end of Section 2.2, the term "locally testable proofs" seems more fitting than the standard term "probabilistically checkable proofs" (abbreviated PCPs), 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.

4.1 Historical notes

The celebrated story of the PCP Theorem is well-known; still we provide a brief overview and refer the interested reader to the account in [32, Sec. 2.6.2] (partially reproduced in the chapter notes

[57] We mention that, due to a technical difficulty, it is easier to establish the claimed bound of $\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1))$ rather than $\Omega(t \cdot \mathtt{vlt}(G_1, \Phi_1))$.

[58] The PCP-system referred to here has arbitrary soundness error (i.e., it rejects the instance (G_2, Φ_2) with probability $\mathtt{vlt}(G_2, \Phi_2) \in [0,1]$).

of [34, Chap. 9]).

The *PCP model* was suggested by Fortnow, Rompel, and Sipser [29] as a model capturing the power of the (related) model of multi-prover interactive proofs, which was suggested by Ben-Or, Goldwasser, Kilian and Wigderson [10] as a generalization of the model of interactive proofs (suggested by Goldwasser, Micali and Rackoff [42]).

The *PCP Theorem* itself is a culmination of a sequence of works, starting with Babai, Fortnow, and Lund [5], who showed that (unrestricted) PCPs (which are merely restricted by the verification time) captured the class $\mathcal{NEXP}$, continuing with the different "scale downs"[59] of that result to the poly-logarithmic query complexity level (by Babai, Fortnow, Levin and Szegedy [6] and Feige, Goldwasser, Lovász, Safra and Szegedy [27]), and culminating with the PCP characterizations of $\mathcal{NP}$ (by Arora and Safra [4] and Arora, Lund, Motwani, Sudan and Szegedy [3]). These developments were inspired by the discovery of the power of interactive proof systems and made use of techniques developed towards this end (by Lund, Fortnow, Karloff, Nisan, and Shamir [53, 62]). The alternative proof of the PCP Theorem was found by Dinur [22] more than a decade later.

The model of *PCPs of Proximity* was introduced by Ben-Sasson, Goldreich, Harsha, Sudan and Vadhan [11], and is almost identical to the notion of Assignment Testers introduced independently by Dinur and Reingold [24].[60] We believe that the proof composition paradigm (of [4]) becomes more clear when explicitly referring to the inner verifiers as PCPs of Proximity (and to the outer verifiers as being robust). In retrospect, the work of [6] should be viewed as a PCP of Proximity of poly-logarithmic verification time for statements that are encoded using a specific error correction code.

There is a fair amount of confusion regarding credits for the introduction of the notion of *locally testable codes* (LTCs). This definition (or at least a related notion)[61] is arguably implicit in [6] as well as in subsequent works on PCP. However, as discussed in Section 2.4, these implicit definitions do not differentiate between the actual notion and related ones (see, e.g., Footnote 61). The definition of locally testable codes has appeared independently in the works of Friedl and Sudan [30] and Rubinfeld and Sudan [61] as well as in the PhD Thesis of Arora [2]. The distinction between the weak and strong notions (see Definition 7) is due to Goldreich and Sudan [41], which initiated a systematic study of these notions.

As stated in Section 3.2.2, our presentation reverses the historical order in which the corresponding results (for codes and proofs) were achieved. That is, the constructions of locally testable proofs of polynomial length, captured in the PCP Theorem [4, 3], predated the coding counterparts.

4.2 On obtaining super-fast testers

Our motivation for studying locally testable codes and proofs referred to super-fast testing, but our actual definitions have focused on the query complexity of these testers. While the query complexity of testing has a natural appeal, the hope is that low query complexity testers would also yield super-fast testing. Indeed, in the case of codes, it is typically the case that the testing time

[59] The term "scale down" is meant to capture the conceptual contents of moving from $\mathcal{NEXP}$ to $\mathcal{NP}$. It is certainly not meant to diminish the impressive technical achievement involved.

[60] Both notions are (important) special cases of the general definition of a "PCP spot-checker" formulated before by Ergün *et al.* [26].

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

is related to the query complexity. However, in the case of proofs there is a seemingly unavoidable (linear) dependence of the verification time on the input length. This (linear) dependence can be avoided if one considers PCP of Proximity (see Section 2.3.3) rather than standard PCP. But even in this case, additional work is needed in order to derive testers that work is sub-linear time. The interested reader is referred to [12, 55].

4.3 The alternative regime: LTCs of linear length

It is quite conceivable that there is a trade-off between the level of locality (i.e., number of queries) and length of the object being tester (i.e., code or proof). At least, the currently known results exhibit such a trade-off.

As stated upfront, we have focused on one extreme of the query-vs-length trade-off: We have insisted on a constant number of queries and sought to minimize the length of the code (or proof). The opposite extreme is to insist on codes (or proofs) of linear length, and to seek to minimize the number of queries. In the case of codes, this regime was recently studied by Kopparty, Meir, Ron-Zewi, and Saraf [50, 51], who obtained codes of optimal rate (with respect to their distance) that can be tested using quasi-poly-logarithmically number of queries. Specifically, for any constant $\delta, \eta > 0$ and a sufficiently large finite set Σ, they obtain codes from Σ^k to Σ^n, where $k = (1-\delta-\eta) \cdot n$, that have relative distance δ and can be tested using $(\log k)^{O(\log \log k)}$ queries. We briefly review their ideas next.[62]

A warm-up: the prior state-of-art. For every constant $c > 0$, a folklore construction, which may be traced to [6], achieves a code of constant rate (i.e., $n = O(k)$) that can be tested using k^c queries. For any selected constant $m \in \mathbb{N}$, the construction identifies $[k]$ with H^m, and uses a finite field $\mathcal{F}$ of size $O(|H|)$. The code maps m-variate functions $f : H^m \to \{0,1\}$ to their low degree extension; that is, f is mapped to the polynomial $p : \mathcal{F}^m \to \mathcal{F}$ of individual degree $|H|-1$ that agrees with f on H^m. This code has relative distance $1 - \frac{m \cdot (|H|-1)}{|\mathcal{F}|} > \frac{1}{2}$ rate $(|H|/|\mathcal{F}|)^m = \exp(-O(m))$, and it can be checked by inspecting the values of the purported codeword $w : \mathcal{F}^m \to \mathcal{F}$ on $O(m)$ random axis-parallel lines, which means making $O(m) \cdot |\mathcal{F}| = O(m) \cdot k^{1/m}$ queries. (A binary code can be obtained by encoding the symbols of $\mathcal{F}$ via a good binary error correcting code.)

Note that we can use the foregoing construction with $m = 1$ and $|F| = (1 + \eta(k)) \cdot k$, for a vanishing function η (e.g., $\eta(k) = 1/\sqrt{k}$). In this case, we obtain a code with very low relative distance (i.e., the relative distance is $\eta(k)/(1 + \eta(k)) \approx \eta(k)$) such that testing is performed by reading the entire purported codeword. Still, such trivial codes (which have very poor distance but very high rate) will be an ingredient in the following construction. A crucial tool that allows their usage is distance amplification, which is actually the pivot of the entire construction.

Distance amplification. Our aim here is to amplify the relative distance of locally testable codes while preserving their local testability. Specifically, starting with a code of relative distance δ, we can obtain a code of any desired relative distance $\delta' \in (0,1)$, while increasing the query complexity (of codeword testing) by a factor of $\text{poly}(1/\delta)$, and decreasing the rate by only a factor of approximately $1-\delta'$. Denoting the initial code by $C : \Sigma^\ell \to \Sigma^m$, and using an auxiliary encoding

[62] Our presentation uses extracts from [51, Sec. 1.2], and we thank the authors for the permission to use these extracts. We omit the credits for various ingredients of the construction, and refer the interested reader to [51, Sec. 1.2].

$E : \Sigma \to \Lambda^t$ and a permutation $\pi : [m \cdot t] \to [m \cdot t]$, we derive a code $C' : \Sigma^\ell \to \Xi^m$, where $\Xi \equiv \Lambda^t$. The codeword $C'(x)$ is obtained by encoding each of the m symbols of $y_1 \cdot y_2 \cdots y_m = C(x)$ via E, permuting the resulting $m \cdot t$-long sequence $E(y_1) \cdot E(y_2) \cdots E(y_m) \in (\Lambda^t)^m$ according to π, and viewing each block of t consecutive Ξ-symbols (in the result) as a symbol of Ξ. That is, the i^{th} symbol of $C'(x)$ equal the t-tuple

$$((E(y_1) \cdot E(y_2) \cdots E(y_m))_{\pi((i-1)\cdot t+1)}, ..., (E(y_1) \cdot E(y_2) \cdots E(y_m))_{\pi(i\cdot t)}) \tag{8}$$

$$\text{where } y_1 \cdot y_2 \cdots y_m = C(x). \tag{9}$$

If E is a very good code, which we can afford given that Σ is relatively small, and π is "sufficiently random" (e.g., the permutation defined by the edge-set of a t-regular m-vertex expander will do)[63], then this construction amplifies distances (see next) although the alphabet size is increased (which is an issue that we already dealt with in other parts of this text). To see why the distance is increased by Eq. (8), recall that for any $x \neq y$ the codewords $C(x)$ and $C(y)$ are at relative distance δ, and the encoding E preserves this distance up to a constant factor (when considering the result as a sequence over Λ). But π "distributes" these $\Omega(\delta \cdot mt)$ symbols among almost all m symbols of Ξ, which yields the claimed relative distance of $\delta' < 1$. Note that if π were totally random, then using $t = O(1/\delta)$ will do (for a fixed pair $x \neq y$), and it can be shown that $t = \text{poly}(1/\delta)$ suffices when using an expander (as outlined in Footnote 63).

The iterative construction. With these preliminaries in place, we turn to the heart of the construction (of [51]), which is an iterative process. The process starts with a code of very small length, which can be tested simply by reading the entire purported codeword. Then, the length is increased iteratively, while the rate, relative distance, and query complexity (of codeword testing) are not harmed too much. Specifically, when wishing to obtain a code with length n, we start with a code of length $\text{poly}(\log n)$, rate $1 - (1/\text{poly}(\log n))$, and relative distance $1/\text{poly}(\log n)$. In each iteration (to be described next), the length and the rate are (roughly) squared, the relative distance is maintained, and the query complexity is increased by a factor of $\text{poly}(\log n)$. Thus, after approximately $\log \log n$ iterations, we obtain a code of length n, constant rate (since $(1 - (1/\text{poly}(\log n)))^{2^{\log \log n}} = 1 - o(1)$), relative distance $1/\text{poly}(\log n)$, and query complexity $(\log n)^{O(\log \log n)}$. Using the foregoing distance-amplification (which is also used inside the iterative process), this gives a code of high rate with constant relative distance and query complexity $(\log n)^{O(\log \log n)}$, as asserted upfront.

A single iteration. Suppose that iteration i begins with a code C_i that has has length n_i, rate r_i, relative distance δ_i, and query complexity q_i. The iteration consists of two steps.

- *Tensor product*: First, we take the tensor product of C_i, denoted C_i^2, where the tensor product of C_i consists of all $n_i \times n_i$ matrices such that that each of the rows and columns of the matrix is a codeword of C_i. The code C_i^2 has length n_i^2, rate r_i^2, and relative distance δ_i^2. Using additional features of the code C_i, which are preserved in the iterations, one can show that C_i^2 is testable with query complexity $q_i \cdot \text{poly}(1/\delta_i)$.

[63] Specifically, we consider the permutation that maps (u, i) to (v, j) if $\{u, v\}$ is the i^{th} edge incident at u and the j^{th} edge incident at v.

- *Distance amplification*: Next, we apply the foregoing distance-amplification to the code C_i^2, and amplify the relative distance from δ_i^2 to δ_i. The resulting code has length $n_{i+1} = O(n_i^2)$, relative distance $\delta_{i+1} = \delta_i = \delta_1$, rate $r_{i+1} = (1-\delta_1) \cdot r_i^2 = (1-delta_1)^{2^i-1} \cdot r_1^{2^i}$, and query complexity $q_{i+1} = q_i \cdot \mathrm{poly}(1/\delta_1)$.

Indeed, a crucial detail that we refrained from addressing is the testing of the tensor code. The interested reader is referred to [51].

4.4 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).[64] 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 strictly sub-exponential length. Specifically, k information bits can be encoded by codewords of length $n = \exp(k^{o(1)})$ that are locally decodable using three bit-probes (cf. [25], building over [67]). It is also known that locally testable codes cannot be T2-nearly linear: Recovery based on q queries requires length at least $k^{1+(2/(q-1))}$ (cf. [46, 48]). Indeed, the gap between the known upper and lower bounds is huge. (We mention that locally decodable codes are related to schemes of (information theoretic) Private Information Retrieval, introduced in [21].)

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 special failure 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 say "`don't know`". This relaxed notion of local decodability can be supported by codes that have length $\ell(k) = k^c$ for any constant $c > 1$ (cf. [11, Sec. 4.2]).[65]

An obvious open problem is to separate locally decodable codes from relaxed locally decodable codes (or to refute this conjectured separation). This separation may follow if one establishes a $k^{1+\ell(q)}$ lower bound on the length of q-query locally decodable codes and a $k^{i+u(k)}$ upper bound on the length of the relaxed counterparts such that $\ell(q) > u(q)$, but currently we have $\ell(q) = 2/(q-1)$ and $u(q) = O(1/\sqrt{q})$. A more ambitious goal is to determine whether there exist locally decodable codes of polynomial length.

4.5 Exercises

Exercise 1 (ϵ-testing a code of relative distance 0.99ϵ): *Show that for every $\epsilon \in (0, 0.001)$ there exists a code of relative distance 0.99ϵ and constant rate that can be ϵ-tested with $O(1/\epsilon)$ queries.*

[64] The aim in this case is to minimize the length of the code. A dual regime refers to allowing only linear length codes and minimizing the query complexity (cf. [50]).

[65] That is, relaxed locally decodable codes of T1-nearly linear length are known [11]. In contrast, by [46], there exist no (non-relaxed) locally decodable codes of T2-nearly linear length.

Guideline: We start with any code $C_0 \subset \{0,1\}^n$ of constant rate, distance $d = 0.499n$ and covering radius smaller than d. (Such a code can be obtained by iteratively adding to the code any n-bit string that is at distance at least d from the current code.) Now, consider the code $C(x) = C_0(x) \cdot 0^m$, and note that C has relative distance greater than $d/(n+m)$ whereas any string that is ϵ-far from C contains at least $\epsilon \cdot (n+m) - d$ non-zeros it its m-bit long suffix.[66] Letting $m = 1.001d/\epsilon$ and using $d = 0.499n$, the claim follows.

Exercise 2 (from a PCP of Proximity for a set of codewords to a PCP for the corresponding decodings): *Suppose that C is an efficiently computable code of relative constant distance greater than ϵ and constant rate ρ, and that V is a PCP of Proximity for $T \stackrel{\text{def}}{=} \{C(x) : x \in S\}$ with proximity parameter ϵ, query complexity q and proof complexity p. Present a PCP for S with query complexity $q'(n) = q(n/\rho)$ and proof complexity $p'(n) = p(n/\rho)$.*

Guideline: On input x and access to an alleged proof π, the verifier computes $w = C(x)$, and invokes $V^{w,\pi}(1^{|w|})$; that is, the verifier emulates queries to w by itself and answers queries to π by querying its own oracle. The point is that if $x \notin S$, then $C(x)$ is ϵ-far from T.

Exercise 3 (from PCP of Proximity to a locally testable code): *Let $\mathcal{C}_0 \subset \{0,1\}^n$ be an efficiently computable code of constant rate. Suppose that V is a PCP of Proximity for the set of all C_0-codewords with proximity parameter ϵ, query complexity q, and proof complexity p. Construct a q-locally 3ϵ-testable code of length $O(p(n)/\epsilon)$ and relative distance that approximately equals that of $\mathcal{C}_0$.*

Guideline: Let $\mathcal{C}(x) = \mathcal{C}_0(x)^t \pi(x)$, where $\pi(x)$ is the (canonical) proof that $\mathcal{C}_0(x)$ is a codeword of $\mathcal{C}_0$, and $t = O(\epsilon^{-1}|\pi(x)|/|\mathcal{C}_0(x)|)$. On input $w^{(1)} \cdots w^{(t)} \pi$, the 3ϵ-tester for $\mathcal{C}$ checks that $w^{(1)} \cdots w^{(t)}$ consists of t repetitions of the n-bit string $w^{(1)}$, and invokes V while proving it access to $w^{(1)}$ (as main input) and to π (as an alleged proof). The key observation is that if $w^{(1)} \cdots w^{(t)} \pi$ is 3ϵ-far from C, then $w^{(1)} \cdots w^{(t)}$ is 2ϵ-far from C_0^t. Hence, either $w^{(1)} \cdots w^{(t)}$ is ϵ-far from $w^{(1)} \cdots w^{(1)}$ or $w^{(1)}$ is ϵ-far from S.

Exercise 4 (interactive proofs yield PCPs): *Suppose that S has an interactive proof system in which the prover sends b bits. Show that S has a PCP of query complexity b.*

Guideline: The queries correspond to possible partial transcripts of the interaction of the verifier with the prover.

Exercise 5 (Satisfiability of Quadratic Systems over GF(2)): *Prove that the following problem is NP-complete. An instance of the problem consists of a system of quadratic equations over* GF(2)*, and the problem is to determine whether there exists an assignment that satisfies all the equations.*

Guideline: Start by showing that the corresponding problem for cubic equations is NP-complete, by a reduction from 3SAT that maps the clause $x \vee \neg y \vee z$ to the equation $(1-x) \cdot y \cdot (1-z) = 0$. Reduce the problem for cubic equations to the problem for quadratic equations by introducing auxiliary variables; that is, given an instance with variables $x_1, ..., x_n$, introduce the auxiliary variables $x_{i,j}$'s and add equations of the form $x_{i,j} = x_i \cdot x_j$.

[66] Hence, the density of non-zeros in this suffix is greater than $\epsilon - (d/m)$.

Exercise 6 (on testing equality of matrices): *Prove that for Boolean n-by-n matrices $A \neq B$, when $r, s \in \{0,1\}^n$ are uniformly selected vectors, it holds that $\mathbf{Pr}_s[As = Bs] = 2^{-\mathrm{rank}(A-B)}$ and it follows that $\mathbf{Pr}_{r,s}[rAs = rBs] \leq 3/4$.*

Guideline: The second assertion follows from the first one by observing that if $(u_1, ..., u_n) \neq (v_1, ..., v_n) \in \{0,1\}^n$, then $\mathbf{Pr}_r[\sum_i r_i u_i = \sum_i r_i v_i] = 1/2$, when $r = (r_1, ..., r_n)$ is uniformly distributed in $\{0,1\}^n$. The first assertion is proved by a generalization of the latter argument.[67]

Exercise 7 (a trivial PCP with large soundness error): *Present a three-query PCP of logarithmic radomness complexity and soundness error $1 - (1/m)$ for* `3SAT`*, where m denotes the number of clauses.*

Guideline: The proof-oracle will be viewed of a truth assignment to the input formula.

Exercise 8 (on the complexity of $\mathtt{gapCSP}_c^{\{0,1\}}$): *Show that for every function $\tau : \mathbb{N} \to (0,1]$, the problem $\mathtt{gapCSP}_\tau^{\{0,1\}}$ is solvable in polynomial-time.*

Guideline: Reduce solving $\mathtt{gapCSP}_\tau^{\{0,1\}}$ to deciding the satisfiability of `2CNF` formulae.

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*, Springer LNCS, Vol. 2764, pages 188–199, 2003.

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

[3] 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).

[4] 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).

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

[6] 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.

[7] 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.

[67] To analyze $\mathbf{Pr}_r[\sum_i r_i u_i = \sum_i r_i v_i] = 1/2$, consider $(w_1, ..., w_n) = (u_1, ..., u_n) - (v_1, ..., v_n)$, and show that $\mathbf{Pr}_r[\sum_i r_i w_i = 0] = 1/2$, by observing that $\sum_{i \in [n]} r_i w_i = \sum_{i:w_i=1} r_i$. Similarly, prove that $\mathbf{Pr}_s[Ds = 0] = 2^{-\mathrm{rank}(D)}$, by showing that for any full rank k-by-k submatrix D' and any $v' \in \{0,1\}^k$ it holds that $\mathbf{Pr}_{s'}[D's' = v'] = 2^{-k}$.

[8] 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.

[9] 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).

[10] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability. In *20th ACM Symposium on the Theory of Computing*, pages 113–131, 1988.

[11] 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.

[12] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan and S. Vadhan. Short PCPs verifiable in polylogarithmic time. In *20th IEEE Conference on Computational Complexity*, pages 120–134, 2005.

[13] 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.

[14] E. Ben-Sasson, V. Guruswami, T. Kaufman, M. Sudan and M. Viderman. Locally testable codes require redundant testers. In *24th IEEE Conference on Computational Complexity*, pages 52–61, 2009.

[15] 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.

[16] E. Ben-Sasson and M. Sudan. Robust locally testable codes and products of codes. In Proceedings of *Random-Approx'04*, Springer LNCS Vol. 3122, pages 286–297, 2004. See ECCC TR04-046, 2004.

[17] E. Ben-Sasson and M. Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, Vol. 38 (2), pages 551–607, 2008. (Preliminary Version in *37th STOC*, 2005).

[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*, Vol. 47 (3), pages 549–595, 1993. (Preliminary Version in *22nd STOC*, 1990).

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

[21] 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.

[22] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, Vol. 54 (3), Art. 12, 2007. Extended abstract in *38th STOC*, 2006.

[23] I. Dinur and P. Harsha. Composition of low-error 2-query PCPs using decodable PCPs. In *50th IEEE Symposium on Foundations of Computer Science*, pages 472–481, 2009.

[24] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. *SIAM Journal on Computing*, Vol. 36 (4), pages 975–1024, 2006. Extended abstract in *45th FOCS*, 2004.

[25] K. Efremenko. 3-query locally decodable codes of subexponential length. In *41st ACM Symposium on the Theory of Computing*, pages 39–44, 2009.

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

[27] 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).

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

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

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

[31] 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.

[32] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998.

[33] O. Goldreich. Short locally testable codes and proofs (survey). ECCC Technical Report TR05-014, Jan. 2005.

[34] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[35] O. Goldreich. Short locally testable codes and proofs: A survey in two parts. In [36].

[36] O. Goldreich (ed.). *Property Testing: Current Research and Surveys*. Springer, LNCS, Vol. 6390, 2010.

[37] 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).

[38] O. Goldreich, T. Gur, and I. Komargodski. Strong Locally Testable Codes with Relaxed Local Decoders. In *30th* IEEE Conference on Computational Complexity, pages 1–41, 2015.

[39] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 3, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[40] O. Goldreich and D. Ron. On proximity oblivious testing. *ECCC*, TR08-041, 2008. Also in the proceedings of the *41st STOC*, 2009.

[41] 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, pages 13–22. (See ECCC Report TR02-050, 2002).

[42] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.

[43] T. Gur and R. Rothblum. Non-interactive proofs of proximity. Technical Report TR13-078, ECCC, 2013.

[44] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica 182* (1999), 105–142. (Preliminary Versions in *28th STOC*, 1996, and *37th FOCS*, 1997).

[45] J. Håstad. Some optimal inapproximability results. *Journal of the ACM 48*, 4 (July 2001), 798–859. (Preliminary Version in *29th STOC*, 1997).

[46] 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*, pages 80–86, 2000.

[47] T. Kaufman, S. Litsyn, and N. Xie. Breaking the ϵ-soundness bound of the linearity test over GF(2). *SIAM Journal on Computing*, Vol. 39 (5), pages 1988–2003, 2009/2010.

[48] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Science*, Vol. 69 (3), pages 395–420, 2004. (Preliminary Version in *35th STOC*, 2003).

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

[50] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf. High rate locally-correctable and locally-testable codes with sub-polynomial query complexity *ECCC*, TR15-068, 2015.

[51] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf. High-rate Locally-testable Codes with Quasi-polylogarithmic Query Complexity. *ECCC*, TR15-110, 2015.

[52] D. Lapidot and A. Shamir. Fully parallelized multi prover protocols for NEXP-time. In *32nd IEEE Symposium on Foundations of Computer Science*, Oct. 1991, pages 13–18.

[53] 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.

[54] O. Meir. Combinatorial construction of locally testable codes. *SIAM Journal on Computing*, Vol. 39 (2), pages 491–544, 2009. Extended abstract in *40th STOC*, 2008.

[55] O. Meir. Combinatorial PCPs with efficient verifiers. In *50th IEEE Symposium on Foundations of Computer Science*, pages 463–471, 2009

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

[57] D. Moshkovitz and R. Raz. Two query PCP with sub-constant error. In *49th IEEE Symposium on Foundations of Computer Science*, pages 314–323, 2008.

[58] I. Newman. Property Testing of Massively Parametrized Problems – A Survey. In [36].

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

[60] R. Raz. A parallel repetition theorem. *SIAM Journal of Computing 27*, 3 (June 1998), 763–803. (Preliminary Version in *27th STOC*, 1995.)

[61] 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).

[62] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[63] D. Spielman. *Computationally efficient error-correcting codes and holographic proofs*. PhD thesis, Massachusetts Institute of Technology, June 1995.

[64] 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.

[65] M. Viderman. Strong LTCs with inverse poly-log rate and constant soundness. In the *54th IEEE Symposium on Foundations of Computer Science*, pages 330–339, 2013.

[66] M. Viderman. Explicit strong LTCs with inverse poly-log rate and constant soundness. *ECCC*, TR15-020, 2015.

[67] S. Yekhanin. Towards 3-Query locally decodable codes of subexponential length. In *39th ACM Symposium on the Theory of Computing*, pages 266–274, 2007.