Relaxed Locally Correctable Codes*

Tom Gur[†1], Govind Ramnarayan[‡2], and Ron D. Rothblum[§3]

1 **UC Berkeley, CA, USA**
`tom.gur@berkeley.edu`
2 **MIT, MA, USA**
`govind@mit.edu`
3 **MIT and Northeastern University, MA, USA**
`ronr@csail.mit.edu`

Abstract

Locally decodable codes (LDCs) and locally correctable codes (LCCs) are error-correcting codes in which individual bits of the message and codeword, respectively, can be recovered by querying only few bits from a noisy codeword. These codes have found numerous applications both in theory and in practice.

A natural relaxation of LDCs, introduced by Ben-Sasson *et al.* (SICOMP, 2006), allows the decoder to reject (i.e., refuse to answer) in case it detects that the codeword is corrupt. They call such a decoder a *relaxed decoder* and construct a constant-query relaxed LDC with almost-linear blocklength, which is sub-exponentially better than what is known for (full-fledged) LDCs in the constant-query regime.

We consider an analogous relaxation for local *correction*. Thus, a *relaxed local corrector* reads only few bits from a (possibly) corrupt codeword and either recovers the desired bit of the codeword, or rejects in case it detects a corruption.

We give two constructions of relaxed LCCs in two regimes, where the first optimizes the query complexity and the second optimizes the rate:

1. **Constant Query Complexity:** A relaxed LCC with polynomial blocklength whose corrector only reads a constant number of bits of the codeword. This is a sub-exponential improvement over the best *constant query* (full-fledged) LCCs that are known.
2. **Constant Rate:** A relaxed LCC with *constant rate* (i.e., linear blocklength) with quasi-polylogarithmic query complexity (i.e., $(\log n)^{O(\log \log n)}$). This is a nearly sub-exponential improvement over the query complexity of a recent (full-fledged) constant-rate LCC of Kopparty *et al.* (STOC, 2016).

1998 ACM Subject Classification F.1.3 Computation by Abstract Devices, Complexity Measures and Classes

Keywords and phrases Coding Theory, Locally Correctable Codes, Probabilistically Checkable Proofs

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.27

* A full version of the paper is available at [15], `https://eccc.weizmann.ac.il/report/2017/143/`

† Tom Gur is partially supported by the UC Berkeley Center for Long-Term Cybersecurity, the ISF grant number 671/13, and Irit Dinur's ERC grant number 239985.
‡ Govind Ramnarayan is supported by National Science Foundation grant number 1218547.
§ Ron D. Rothblum is partially supported by NSF MACS - CNS-1413920, SIMONS Investigator award Agreement Dated 6-5-12 and the Cybersecurity and Privacy Institute at Northeastern University.

1 Introduction

Dating back to the seminal works of Shannon [24] and Hamming [17], error correcting codes are used to reliably transmit data over noisy channels and store data. Roughly speaking, error correcting codes are injective functions that take a message and output a codeword, in which the message is encoded with extra redundancy, with the property that even if some of the symbols in the codeword are corrupted, the message is still recoverable.

Locally decodable codes (LDCs) and *locally correctable codes* (LCCs) are error correcting codes that admit highly efficient procedures for recovering small amounts of data. More specifically, in an LDC, a single symbol of the message can be recovered by only reading a few bits from a noisy codeword. An LCC has the same property but with respect to recovering bits of the *codeword* itself (rather than the message).

Locally decodable codes and locally correctable codes have had a profound impact various areas of theoretical computer science including cryptography, PCPs, hardness of approximation, interactive proofs, private information retrieval, program checking, and databases (see [26] and the more recent [20] for a survey on local decodable and correctable codes). While these codes have found numerous uses in theory and practice, one significant downside is that current constructions require adding a large amount of redundancy. Specifically, to decode or correct with a *constant* number of queries, the current state of the art LDCs have super polynomial blocklength [25, 10] (by blocklength we refer to the length of the codeword as a function of the message length) and the current best LCC, which has *sub-exponential* blocklength.[1]

Motivated by this, Ben-Sasson *et al.* [5] defined a natural relaxation of locally decoding, for which they could achieve a dramatically better blocklength. Roughly speaking, their relaxation allows the decoder to abort in case of failure, while still requiring the decoder to successfully decode non-corrupted codewords (in particular, this prevents the decoder from always aborting). Moreover, in the constant query regime, such codes can be transformed to codes, with similar parameters, that are guaranteed to successfully decode on the majority of message bits.

Thus, a *relaxed local decoder* for a code C gets oracle access to a string w that is relatively close to some codeword $c = C(x)$ and an index $i \in [|x|]$. The decoder should make only few queries to w and satisfy the following:

1. If the string $w = c$ (i.e., w is an uncorrupted codeword), the relaxed decoder must always output x_i.
2. Otherwise, with high probability, the decoder should either output x_i or a special "abort" symbol $\perp$ (indicating the decoder detected an error and is unable to decode).[2]

The additional freedom introduced by allowing the decoder to abort turns out to be extremely useful. Using the notion of PCPs of proximity (PCPP), which they also introduce[3] and construct, Ben-Sasson *et al.* obtain relaxed locally decodable codes (RLDCs) with constant query complexity and almost-linear blocklength.

In this work we extend the relaxation of Ben-Sasson *et al.* to LCCs and define the analogous notion of relaxed LCCs as follows: We say that a code $C : \Sigma^k \to \Sigma^n$ is a *relaxed*

[1] These are Reed-Muller codes over a constant-size alphabet and with constant degree (but large dimension).

[2] The actual definition in [5] also requires that for a constant fraction of the coordinates, the decoder decodes correctly (i.e., does not output $\perp$) with high probability. However, they later show that this additional condition follows from Conditions (1) and (2) above. See further discussion in the full version.

[3] The equivalent notion of *assignment tester* was introduced independently by Dinur and Reingold [9].

LCC with query complexity q, if there exists a corrector that has oracle access to a string $w \in \Sigma^n$, which is close to some codeword $c \in C$, and also gets as explicit input an index $i \in [n]$. The algorithm makes at most q queries to the string w, and satisfies the following:

1. If $w = c$ (i.e., w was not corrupted), then the corrector always outputs c_i.
2. Otherwise, with high probability, the corrector either outputs c_i, or a special "abort" symbol $\perp$.

The remarkable savings achieved by Ben-Sasson *et al.* begs the question: can relaxed locally *correctable* codes achieve similar savings in blocklength over current constructions of locally correctable codes? We answer this question in the affirmative by constructing relaxed LCCs with significantly better parameters than that of the state-of-the-art (full-fledged) LCCs.

2 Our Results

In this work, we construct relaxed locally correctable codes in two different parameter regimes: the first, which we view as our main technical contribution, focuses on the constant *query complexity* regime, whereas the second, which is easier to prove given previous work, focuses on constant *rate*.

Constant Query RLCC

Our first result is a relaxed LCC which requires only $O(1)$ queries and has a polynomial blocklength.

▶ **Theorem 1** (Constant Query Relaxed LCC, Informally Stated). *There exists a relaxed* LCC $C : \{0,1\}^k \to \{0,1\}^n$ *with constant relative distance, constant query complexity, and blocklength* $n = \text{poly}(k)$. *Furthermore,* C *is a linear code.*

Theorem 1 yields a sub-exponential improvement compared to the best known (full-fledged) LCCs with constant query complexity, which have sub-exponential blocklength. This result heavily relies on a certain type of PCPs of proximity (PCPPs) that we construct. We elaborate on our PCPP constructions in Section 2.1 below.

We remark that the specific blocklength in Theorem 1 is roughly *quartic* (i.e., fourth power) in the message length. Constructing a constant-query RLCC with a shorter blocklength (let alone an (almost) linear one, as known for relaxed LDCs) is an interesting open problem.

Constant Rate RLCC

Our second main result is a construction of a relaxed LCC with *constant rate*[4] and almost polylogarithmic query complexity.

▶ **Theorem 2** (Constant Rate Relaxed LCC, Informally Stated). *There exists a relaxed* LCC $C : \{0,1\}^k \to \{0,1\}^n$ *with constant relative distance, query complexity* $(\log n)^{O(\log \log n)}$, *and constant rate (i.e., blocklength* $n = O(k)$*). Furthermore,* C *is a linear code and has distance-rate tradeoff approaching the Zyablov bound [27].*

[4] Recall that the rate of a code $C : \Sigma^k \to \Sigma^n$ is defined as k/n. We use the terms "constant rate" and "linear blocklength" interchangeably.

This is a nearly sub-exponential improvement in query complexity over the best constant rate (full-fledged) LCCs, due to Kopparty *et al.* [19], which requires $2^{\tilde{O}(\sqrt{\log n})}$ queries for correction. As a matter of fact, our construction is essentially identical to one of the constructions of [19].[5] Our main insight in proving Theorem 2 is that their code allows for *relaxed* local correction with much better parameters.[6] As a secondary contribution, we also provide a modular presentation for the *distance amplification* step, which is the main step in [19] (and is originally due to Alon, Edmonds and Luby [1]).

▶ Remark. As mentioned in Footnote 2, the original definition of RLDC [5] includes a third condition, which requires that the decoder must successfully decode a constant fraction of the coordinates. More precisely, for every $w \in \Sigma^n$ that is close to some codeword $c = C(x)$, there exists a set $I_w \subseteq [k]$ of size $\Omega(k)$ such that for every $i \in I_w$ with high probability the decoder D outputs x_i (rather than outputting $\perp$).

Ben-Sasson *et al.* showed that every RLDC with constant query complexity that satisfies the first two conditions, can be transformed into an RLDC with similar parameters that satisfies the third condition as well. We remark that this transformation also applies to RLCCs with constant query complexity. However, for super-constant query complexity (as in Theorem 2) the same transformation only guarantees successful decoding of a constant fraction of coordinates, if the codeword is corrupted on a sub-constant fraction of its coordinates (i.e., the fraction roughly corresponds to the reciprocal of the query complexity).

▶ Remark. Both our constant-query and constant-rate RLCCs are systematic[7]. Hence they are automatically also relaxed locally *decodable* codes (i.e., RLDCs). In particular, the code from Theorem 2 is also the first construction of a relaxed locally *decodable* code in the constant-rate regime, with query complexity $(\log n)^{O(\log \log n)}$.

2.1 PCP Constructions

PCPs of proximity (PCPP), first studied by Ben-Sasson *et al.* [5] and by Dinur and Reingold [9] were originally introduced to facilitate PCP composition. Beyond their usefulness in PCP constructions, of PCPPs have proved to be extremely useful in coding theory as well. Indeed, PCPPs lie at the heart of several constructions of LTCs [14], relaxed LDCs [5, 13], universal LTCs [11, 12], as well as in our construction of relaxed LCCs (specifically in Theorem 1).

Loosely speaking, a PCPP is a proof system that allows for probabilistic verification of approximate decision problems by querying only a small number of locations in both the statement and the proof. (In contrast, a standard PCP verifier reads the *entire* statement, and probabilistically verifies an *exact* decision problem, by querying only a small number of locations in the proof.) Similarly to the scenario in property testing, the soundness guarantee provided by PCPPs is that the PCPP verifier is only required to reject statements that are "far" (in Hamming distance) from being correct.

In this work, we provide new constructions of PCPPs that play a crucial role in our constant-query relaxed LCC construction. The PCPPs that we construct are for verifying membership in affine subspaces (rather than general languages in P or NP), since this is all that we need for our RLCC constructions. More specifically, we shall construct PCPPs that

[5] Interestingly, our construction is inspired by the [19] construction of a locally *testable* code, rather than their locally *correctable* code.

[6] We were informed that a similar observation about the [19] code has been made recently and independently in an unpublished work of Hemenway, Ron-Zewi, and Wootters [18].

[7] Recall that a code is systematic if the first part of every codeword is the original message.

are: *linear*, *robust*, *self-correctable*, and admit *strong canonical soundness*. We discuss these properties in more detail next (see full version for precise definitions). We remark that our PCPP construction is inspired by the construction of linear-inner proof-systems (LIPS) by Goldreich and Sudan [14].

Linearity

We call a PCPP proof-system *linear* if it satisfies two conditions. First, the prescribed proof π for any statement x must be a linear function of the statement. Second, to decide whether to accept, the PCPP verifier only checks that the values that it reads from the input and PCPP proof lie in an affine subspace. Put differently, the verifier's decision predicate is computable by a linear circuit. We remark that in the literature [6, 22], the term "linear PCPP" sometimes refers only to the latter of the two requirements but here we also insist that the proof be a linear function of the statement.

We use linearity both to assure that our resulting codes are linear codes, as well as to facilitate composition with other PCPPs. We note that standard PCPs are typically inherently non-linear (since they are designed for general languages in P or in NP). However, in our context we are only trying to verify membership in affine spaces and so it is reasonable to expect to have linear PCPPs.

Robustness

The notion of *robust* PCPPs, introduced by Ben Sasson *et al.* [5], refers to PCPP systems whose verifier, roughly speaking, is not only required to reject statements that are far from valid but also that the local view of the verifier (i.e., the answers to the queries made by verifier) is far from any local view that would have caused the verifier to accept. Robustness plays a key role in enabling PCP composition. While this condition holds trivially for verifiers with constant query complexity, in our construction we will also consider verifiers with super-constant query complexity, for which achieving robustness is non-trivial.

Self-Correctability

In a *self-correctable* PCPP system, the proof oracle admits a local correction procedure that allows for local recovery of individual bits of a moderately corrupted PCPP proof. The self-correctability of the PCPP oracles allows us to include them as part of an RLCC's codeword.

Strong Canonical Soundness

The notion of PCPPs with *strong canonical soundness*, introduced by Goldreich and Sudan [14], requires that correct inputs (i.e., that reside in the target language) have a canonical proof and the PCPP verifier is required to reject "wrong" (i.e., non-canonical) proofs, *even for correct statements*. In more detail, these PCPPs satisfy two additional requirements: (1) *canonicity*: for every true statement there exists a unique canonical proof that the verifier is required to always accept, and (2) *strong canonical soundness*: the verifier is required to reject any pair (x, π) of statement and proof with probability that is roughly proportional to its distance from a true statement and its corresponding canonical proof.

We are now ready to state our results on PCPs of proximity with the aforementioned properties. The first construction has exponential length and constant query complexity,

whereas the second construction, whose proof is significantly more involved, has polynomial length and poly-logarithmic query complexity.

Our first result is a variant of the Hadamard PCPP, with exponential length but constant query complexity.

▶ **Theorem 3** (Informally stated, see full version). *There exists a linear, self-correctable, strong canonical* PCPP *for membership in affine subspaces, with query complexity* $O(1)$ *and exponential length (in the size of the statement).*

Our second result is a variant of the [4] PCP, which has poly-logarithmic query complexity and polynomial length.

▶ **Theorem 4** (Informally stated, see full version). *There exists a linear, self-correctable,* $\Omega(1)$*-robust, strong canonical* PCPP *for membership in affine subspaces, with query complexity* $\text{polylog}(n)$ *and* $\text{poly}(n)$ *length, for statements of length* n*.*

3 Technical Overview

The techniques used for our two constructions are quite different. We first outline the constant-query result, which is more complex, in Section 3.1 and then outline the constant-rate result in Section 3.2.

3.1 Constant-Query Relaxed LCC

The starting point for our construction is the [5] construction of relaxed locally *decodable* codes (RLDC), which we review next.[8] In their construction, each codeword has two parts: the first part provides the distance, and the second enables relaxed local decodability. More specifically, they construct an RLDC C' whose codewords consist of the following two equal-length parts: (1) repetitions of a codeword $C(x)$, where $C : \{0,1\}^k \to \{0,1\}^n$ is some systematic code with constant distance and rate, (2) for every message bit in $C(x)$, they add a PCPP, which is a proof that x_i is indeed the i^{th} bit of $C(x)$.[9]

We remark that the repetitions in the first part of the code are simply meant to ensure distance (as the PCPP proof strings are not necessarily a code with good distance). To decode, the relaxed decoder for C' invokes the PCPP verifier to check that the i'th bit of the first part is indeed $C(x)_i$ and outputs it, unless the verifier rejects, in which case the relaxed decoder may return $\perp$.

Ben-Sasson *et al.* show that this code is indeed a relaxed LDC. However, in general, it will not necessarily be a relaxed LCC. Specifically, it is unclear how to correct bits that are part of the PCPP proof strings. Simply appending even more PCPPs to deal with the original ones will not do since we will also need to correct those. Moreover, it is worth pointing out that even if the PCPP proof strings had some internal self-correction mechanism, this would still not suffice since each PCPP proof string by itself is very short (as compared to the entire codeword) and could therefore be entirely corrupted.

Before proceeding to cope with this difficulty, we first suggest a different perspective on the [5] construction, which is inspired by the highly influential and useful notion of PCP

[8] We describe the simpler variant of the [5] construction, which achieves nearly *quadratic* blocklength. We remark that [5] also present a more involved construction that achieves nearly *linear* blocklength.

[9] Actually, our presentation differs slightly from that of [5]. Their construction contains an additional part that consists of repetitions of the original *message*. However, when using a systematic code C, this addition is not necessary.

composition [3]. Specifically, we think of the [5] construction as a *composition* of the code C, which is trivially locally decodable with n queries, with a constant-query PCPP. This composition yields a *relaxed* LDC with query complexity $O(1)$, at a moderate increase in blocklength (which comes from appending all of the PCPP proof strings).

We shall adopt the composition perspective, and use it to construct a relaxed locally *correctable* code, by introducing a technique for composing a (possibly relaxed) LCC C with a special type of PCP of proximity (PCPP). The result of the composition is a *relaxed* LCC C' which basically inherits the query complexity of the PCPP (and with a moderate overhead in blocklength).

Similarly to the relaxed LDCs of [5], the codewords of C' are constructed by taking repetitions of a codeword of a (possibly relaxed) robust RLCC C and concatenating it with many PCPP proof strings. Specifically, for each set of queries that the relaxed local corrector for C would like to make, we write down a PCPP proof that this set of queries would be answered correctly. We shall refer to the first part of C', which contains repetitions of C, as the *core* of C', and refer to the second as the PCPP part.

Observe that the foregoing approach allows us to locally correct bits of the *core* of C'. The relaxed corrector for the composed RLCC takes the queries made by the old corrector as input, and uses the PCPP verifier to test if the old corrector would have accepted.[10] However, we shall need a more sophisticated machinery to correct the PCPP part of C' (indeed this is exactly the challenge that we faced when trying to follow the [5] approach). This will be achieved by ensuring that the PCPPs that we use have strong properties.

In particular, we shall employ the foregoing composition strategy while using the PCPPs of Section 2.1, which admit canonical proofs, strong canonical soundness, and self-correctability. Recall that a PCPP is said to have strong canonical soundness if every valid input has a *canonical* proof that it accepts, and any pair of statement and proof are rejected with probability proportional to their distance from a true statement and its corresponding canonical proof. In addition, recall that a canonical PCPP is said to be *self-correctable* if the canonical proof strings form a locally correctable code (i.e., if it is possible to locally recover individual bits of a noisy PCPP oracle).

Suppose that we want to correct a bit that lies in the PCPP part of a purported codeword of C'. If this bit is in a PCPP oracle that is not too corrupted, we can simply use the PCPP's self-corrector to recover the bit. However, as pointed out before, this naive attempt to self-correct fails when the *entire* proof string is corrupted. This can easily happen when the proof strings, each of which refers to a single possible query set of the original corrector, are short relative to the size of the entire codeword.

Thus, our main challenge is to detect whether the given PCPP proof string was (possibly entirely) corrupted. We observe that if on the one hand, the PCPP oracle we wish to correct is heavily corrupted, while on the other hand, the statement to which the PCPP refers (i.e., the queries that the corrector for C makes) is *not* heavily corrupted, then the proof is far from the prescribed canonical proof. The strong canonical soundness guarantees that in such case the PCPP verifier will detect the corruption and reject. Thus, we are left with the case that *both* the PCPP oracle and the statement that it refers to are heavily corrupted.

To detect this deviation, we choose a random point in the foregoing statement and read it directly. Since that point is in the core of the code, and we have already described the

[10] Even for this to work, we need to ensure that the original RLCC is *robust*, in the sense that with high probability the corrector's view (i.e., the answers to its queries) are *far* from answers that would make it output an incorrect value. Otherwise, we have no guarantee that the PCPP verifier will see the error.

procedure for correcting in the core, we can also correct this point and compare the corrected value with the symbol that we read directly. Since we have assumed that the statement was heavily corrupted, the value that we read directly will likely be different than what the corrector returns, in which case we can reject.

Equipped with this composition theorem, we can now construct our code. By applying the composition theorem to the low-degree extension code, of suitable parameters, and the PCPP given in Theorem 3, we can already construct a constant-query RLCC with quasi-polynomial blocklength. We note that this is already a significant improvement over the current best (full-fledged) LCCs. However, to obtain polynomial blocklength, we shall perform two compositions with different PCPPs (in direct analogy to the first proof of the PCP theorem [2]).

As in the quasi-polynomial result mentioned above, our starting point is the low-degree extension code. Under a suitable parameterization, this code is known to be a robust (full-fledged) LCC with almost linear blocklength and polylogarithmic query complexity. We shall first compose it with the polynomial length, polylogarithmic query, strong canonical, self-correctable and robust PCPP of Theorem 4. Since the foregoing PCPP is robust, this composition yields a robust RLCC with polynomial blocklength and slightly sub-logarithmic query complexity. Finally, we compose yet again with the exponential length, constant query, strong canonical, self-correctable PCPP from Theorem 3, which yields an RLCC with constant query complexity.

Each of our two composition steps introduces at least a quadratic overhead to the blocklength. This is because our composition of an RLCC with a PCPP appends a PCPP proof-string for every pair (i, ρ) of coordinate i to be corrected from the base code and random string ρ of the underlying corrector with respect to the point i.[11] Since we apply two such composition steps, we get a code with roughly $n \approx k^4$ blocklength.

3.2 Constant-Rate RLCC

For the constant rate construction, we build on the recent breakthrough construction of locally testable[12] and correctable codes of Kopparty *et al.* [19]. Interestingly, we will actually focus on the [19] construction of locally *testable* codes (rather than correctable ones), even though our own goal is to construct (relaxed) locally *correctable* codes.[13]

Kopparty *et al.* construct locally testable codes with query complexity $(\log n)^{O(\log\log(n))}$ by taking an iterative approach, similar to that of Meir [21]. They start off with a code of dimension $\operatorname{poly}\log(n)$ (which is trivially locally testable, by reading the entire codeword) and gradually increase its blocklength, while (almost) preserving the local testability and maintaining the rate of the code close to 1. This amplification step is achieved by combining two transformations on codes:

1. *Code tensoring*: this transformation squares the block-length (which is good since we want to obtain blocklength n) and rate (which is not too bad since our rate is close to 1). The main negative affect is that this transformation also squares the distance.

[11] In contrast, in standard PCP composition, one only appends an inner PCP proof-string for every random string ρ of the outer PCP. Thus, as long as the randomness complexity of the outer PCP is minimal, it is possible to achieve close to constant multiplicative overhead when composing.

[12] Recall that a locally testable code [14] is a code for which one can test, using a sub-linear number of queries, whether a given string is a codeword or far from such.

[13] This may not be surprising, since the notions of relaxed correctability and testability are closely related. In particular, as observed in [11], every RLDC (analogously, RLCC) is roughly equivalent to a code C such that for every coordinate i and value b, the subcode obtained by fixing the i'th bit to b (i.e., $\{C(x) : C(x)_i = b\}$) is locally testable.

2. *Distance amplification*: remarkably, this transformation fixed the loss in distance caused by the tensoring step, without harming the rate or local testability too much.

As noted above, in their work, Kopparty *et al.* also construct a locally correctable code, albeit only with query complexity $2^{\tilde{O}(\sqrt{\log(n)})}$. The reason why their LCC construction does not match the parameters of their LTC construction is that the tensoring step, used in their construction of locally *testable* codes, is not known to preserve local correctability.[14]

Our key observation is that tensoring does preserve *relaxed* local correctability. Recall that the tensor of a code $C : \mathbb{F}^k \to \mathbb{F}^n$ is the code $C^2 : \mathbb{F}^{k^2} \to \mathbb{F}^{n^2}$ that consists of all strings $c \in \mathbb{F}^{n^2}$, viewed as $n \times n$ matrices, that consist of rows and columns that are each codewords of c.

Suppose that C is a (relaxed) LCC with query complexity q. We want to show that C^2 is also a (relaxed) LCC with query complexity roughly q. Let $w \in \mathbb{F}^{n^2}$ be a (possibly) corrupt codeword of C^2. Thus, w which we also view as an $n \times n$ matrix, is close to some codeword $c \in \mathbb{F}^{n^2}$. Given an index $(i, j) \in [n] \times [n]$ to correct, a natural approach is apply the (relaxed) local corrector of C on the i^{th} row of w, with respect to the index j.

If it were the case that the i^{th} row of w were close to the i^{th} row of c, we would be done. However, the i^{th} row of w only constitutes a $1/n$ fraction of w and so it could potentially be entirely corrupt. Let us assume that it is indeed the case that i^{th} rows of c and w (almost) entirely disagree.

To detect that this is the case, our corrector chooses at random a few columns $J \subset [n]$. On the one hand, since the i^{th} rows of w and c disagree almost everywhere, with high probability for some $j' \in J$ it will be the case that $w_{i,j'} \neq c_{i,j'}$. On the other hand, since j' is just a random column, with high probability the j'^{th} columns of w and c are close.

Given this, a natural approach is to have our corrector read the (i, j')-th entries of w for every $j' \in J$, by applying the (relaxed) local corrector of C. In the likely case that it chooses a j' such that the j'^{th} column of w and c are close, with high probability the corrector will either return $c_{i,j'}$ or $\perp$. If our corrector sees $\perp$ it can immediately reject (since this would never happen for an exact codeword). Otherwise, if our corrector sees the value $c_{i,j'}$, it can compare this value with $w_{i,j'}$ (by explicitly reading the (i, j')'th entry of w). By the above analysis, these values will be different (with high probability), in which case our corrector can also reject.

To calculate the overall query complexity of the resulting code, we need to account for the overhead introduced by both the tensoring and distance amplification steps. Assuming that C is (relaxed) locally correctable up to distance δ_R, the tensoring step only increases the query complexity by $O(1/\delta_R)$. Each distance amplification increases the query complexity by roughly a polylog(n) factor. Thus, since we need roughly $\log \log(n)$ iterations to reach blocklength n, the overall query complexity is $(\log n)^{O(\log \log n)}$.

4 Related Works

A similar notion to RLDCs called *Locally Decode/Reject Codes (*LDRC*s)* arose in the beautiful work of Moshkovitz and Raz [23] on constructing two-query PCPs with sub-constant error. These are similar to RLDCs in that they are codes with a decoder that is permitted to reject if it sees errors. However, it is important to note that the two notions differ in a few significant ways and are overall incomparable. First, LDRCs decode a k-tuple of coordinates jointly,

[14] It can be shown that tensoring at most *squares* the query complexity for local correcting. However, the [19] iterative approach cannot afford such an overhead in each iteration.

rather than a single coordinate. Second, LDRCs have a "list-decoding" guarantee – namely, that the decoding agrees with one message in a small list of messages – as opposed to RLDCs, which provide unique decoding (but up to a smaller radius). Finally, LDRCs only need to work with high probability over the choice of k-tuple, while RLDCs must decode or reject with high probability for *every* coordinate. See [23, Section 2] for the formal definition of LDRCs and a comparison to RLDCs.

Another related notion is that of *decodable* PCP*s* (dPCP), first introduced by Dinur and Harsha [8] to the end of obtaining a modular and simpler proof of the the [23] result. A dPCP is a PCP oracle, encoding an NP-witness, which allows for list decoding of individual bits of the NP witness it encodes. Dinur and Harsha provided constructions of such dPCPs as well as a composition theorem for dPCPs.

Additionally, in a recent work, Goldreich and Gur [11] introduced the notion of *universal locally testable codes* (universal-LTC), which can be thought of as generalizing the notion of relaxed LDCs. A universal-LTC $C : \{0,1\}^k \to \{0,1\}^n$ for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ is a code such that for every $i \in [M]$ and $b \in \{0,1\}$, membership in the subcode $\{C(x) : f_i(x) = b\}$ can be locally tested. As was shown in [11], universal-LTCs with respect to the family of dictators functions (i.e., of the form $f(x) = x_i$) are roughly equivalent to RLDCs. We remark that their formulation can be naturally generalized to also capture the notion of RLCC.

Finally, we remark that the relaxed LDCs have been used in the context of interactive proofs of proximity [16] and property testing [7].

Acknowledgements. We thank Oded Goldreich for initiating a discussion that led to this work, for insightful conversions and for his encouragement. The second author would like to also thank Dana Moshkovitz for useful discussions surrounding this work.

References

1 Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 512–519. IEEE, 1995.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901.

4 László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31. ACM, 1991. doi:10.1145/103418.103428.

5 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810.

6 Eli Ben-Sasson, Prahladh Harsha, Oded Lachish, and Arie Matsliah. Sound 3-query pcpps are long. *TOCT*, 1(2):7:1–7:49, 2009. doi:10.1145/1595391.1595394.

7 Clément L. Canonne and Tom Gur. An adaptivity hierarchy theorem for property testing. *Computational Complexity Conference (CCC)*, 2017.

8 Irit Dinur and Prahladh Harsha. Composition of low-error 2-query pcps using decodable pcps. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009,*

October 25-27, 2009, Atlanta, Georgia, USA, pages 472–481. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.8.

9 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006. doi:10.1137/S0097539705446962.

10 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012. doi:10.1137/090772721.

11 Oded Goldreich and Tom Gur. Universal locally testable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:42, 2016. URL: http://eccc.hpi-web.de/report/2016/042.

12 Oded Goldreich and Tom Gur. Universal locally verifiable codes and 3-round interactive proofs of proximity for CSP. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:192, 2016. URL: http://eccc.hpi-web.de/report/2016/192.

13 Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In David Zuckerman, editor, *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, volume 33 of *LIPIcs*, pages 1–41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CCC.2015.1.

14 Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *J. ACM*, 53(4):558–655, 2006. doi:10.1145/1162349.1162351.

15 Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:143, 2017. URL: https://eccc.weizmann.ac.il/report/2017/143.

16 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Computational Complexity*, pages 1–109, 2016. doi:10.1007/s00037-016-0136-9.

17 Richard W Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2):147–160, 1950.

18 Brett Hemenway, Noga Ron-Zewi, and Mary Wootters, 2017. Personal communication.

19 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally-correctable and locally-testable codes with sub-polynomial query complexity. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 202–215. ACM, 2016. doi:10.1145/2897518.2897523.

20 Swastik Kopparty and Shubhangi Saraf. Local testing and decoding of high-rate error-correcting codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:126, 2017. URL: https://eccc.weizmann.ac.il/report/2017/126.

21 Or Meir. Combinatorial construction of locally testable codes. *SIAM J. Comput.*, 39(2):491–544, 2009. doi:10.1137/080729967.

22 Or Meir. Combinatorial pcps with short proofs. *Computational Complexity*, 25(1):1–102, 2016. doi:10.1007/s00037-015-0111-x.

23 Dana Moshkovitz and Ran Raz. Two-query pcp with subconstant error. *Journal of the ACM (JACM)*, 57(5):29, 2010.

24 Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

25 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008. doi:10.1145/1326554.1326555.

26 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012. doi:10.1561/0400000030.

27 Victor Vasilievich Zyablov. An estimate of the complexity of constructing binary linear cascade codes. *Problemy Peredachi Informatsii*, 7(1):5–13, 1971.