Linear Relaxed Locally Decodable and Correctable Codes Do Not Need Adaptivity and Two-Sided Error

Guy Goldberg
Weizmann Institute of Science, Rehovot, Israel

Abstract

Relaxed locally decodable codes (RLDCs) are error-correcting codes in which individual bits of the message can be recovered by querying only a few bits from a noisy codeword. For uncorrupted codewords, and for every bit, the decoder must decode the bit correctly with high probability. However, for a noisy codeword, a relaxed local decoder is allowed to output a "rejection" symbol, indicating that the decoding failed.

We study the power of adaptivity and two-sided error for RLDCs. Our main result is that if the underlying code is linear, *adaptivity and two-sided error do not give any power to relaxed local decoding.* We construct a reduction from adaptive, two-sided error relaxed local decoders to non-adaptive, one-sided error ones. That is, the reduction produces a relaxed local decoder that never errs or rejects if its input is a valid codeword and makes queries based on its internal randomness (and the requested index to decode), independently of the input.

The reduction essentially maintains the query complexity, requiring at most one additional query. For any input, the decoder's error probability increases at most two-fold. Furthermore, assuming the underlying code is in systematic form, where the original message is embedded as the first bits of its encoding, the reduction also conserves both the code itself and its rate and distance properties

We base the reduction on our new notion of *additive* promise problems. A promise problem is additive if the sum of any two YES-instances is a YES-instance and the sum of any NO-instance and a YES-instance is a NO-instance. This novel framework captures both linear RLDCs and property testing (of linear properties), despite their significant differences.

We prove that in general, algorithms for *any* additive promise problem do not gain power from adaptivity or two-sided error, and obtain the result for RLDCs as a special case. The result also holds for relaxed locally *correctable* codes (RLCCs), where a *codeword* bit should be recovered.

As an application, we improve the best known lower bound for linear adaptive RLDCs. Specifically, we prove that such codes require block length of $n \geq k^{1+\Omega(1/q^2)}$, where k denotes the message length and q denotes the number of queries.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Locally decodable codes, Relaxed locally correctable codes, Relaxed locally decodable codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.74

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2023/067/

Funding *Guy Goldberg*: Research supported by the European Research Council (ERC) CoG (grant No. 772839) and the Israel Science Foundation (grant No. 2073/21).

Acknowledgements We would like to thank Irit Dinur for her guidance and encouragement. We would also like to thank Oded Goldreich for insightful discussions, and to Yotam Dikstein for his helpful comments.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 74; pp. 74:1–74:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Suppose you receive a binary string (a "message"), and would like to know the value of the message at some index i. How many queries do you need to make? The answer is obviously 1, as you can query index i and get the value. But what if some of the message bits are corrupted, because they were, say, transmitted over a noisy channel? The bit at the needed index i might have been corrupted.

Error-correcting codes might help. Such codes allow encoding the message with extra redundancy as a codeword, and the original message can be recovered even if some bits of the codeword were corrupted. However, one needs to read the entire codeword to recover the original message. As our goal was to read only one bit from the message, this solution seems inefficient.

Locally decodable codes (LDCs), introduced by Katz and Trevisan [17], are aimed at solving this problem. These codes are equipped with a local decoding algorithm ("decoder") that recovers each message bit by querying a few bits from a codeword, instead of reading all of it. Two main measures of efficiency for LDCs are the query complexity of the decoder (which we want to be as small as possible) and the rate of the code (which we want to be high). A similar notion, originated in works on program checking by Blum and Kannan [5] and Lipton [21], is of locally *correctable* codes (LCCs). These are error-correcting codes that admit a local algorithm (now called "corrector") that not only recovers each message bit, but is also required to correct any bit from the *codeword*.

LDCs and LCCs have profoundly impacted theoretical computer science and found numerous applications. Despite the extensive research, current constructions require adding a large amount of redundancy. Motivated by this, Ben-Sasson et. al. [3] introduced Relaxed Locally Decodable Codes (RLDCs). For uncorrupted codewords, and for every bit, a relaxed local decoder still must correctly decode the bit with high probability. However, for a noisy codeword, it is now allowed to output a "rejection" symbol, indicating that the decoding failed. This relaxation allows constructing codes with a dramatically better tradeoff between query complexity and rate. Such codes have found various applications, notably in the construction of proof systems (e.g., [22]) and within the area of property testing (e.g., [11]).

More formally, a relaxed local decoder of radius $\rho > 0$ for a code C with soundness error $\epsilon_{\text{soundness}}$ and completeness error $\epsilon_{\text{completeness}}$ is a procedure that gets oracle access to $w \in \{0,1\}^n$, that is ρ-close[1] to some codeword $c = C(x)$ and an index i, and satisfies the following two requirements:

1. (completeness) If w is a valid codeword (that is, $w = c$) then for every i the relaxed local decoder outputs x_i with probability at least $1 - \epsilon_{\text{completeness}}$.
2. (relaxed local decoding) Otherwise, with probability at least $1 - \epsilon_{\text{soundness}}$, the relaxed local decoder outputs x_i or a "reject" symbol $\perp$, indicating the decoding failed.

We call n the *block length* of the code, and the length of x the *message length*.

Gur, Ramnarayan, and Rothblum [14] considered an analogous relaxation for local *correction*, where the corrector either recovers the desired codeword bit, or rejects in case it detects a corruption. They named such codes Relaxed Locally *Correctable* Codes, or RLCCs (see Definition 7).

In this work, we study the power of adaptivity and two-sided error of linear RLDCs and RLCCs. We say that a local algorithm is *adaptive* if it is allowed to choose its queries according to answers of previous queries, and that it is *non-adaptive* otherwise (i.e, if it

[1] Two strings are ρ-close to each other if the normalized Hamming distance between them is at most ρ.

determines its queries based only on its internal coin tosses). Though adaptive algorithms are syntactically stronger than non-adaptive ones, all known constructions of RLDCs and RLCCs are non-adaptive (e.g. [3, 15, 2, 7, 20, 8]). This raises the question of whether it is possible to cleverly utilize adaptivity in order to make improved constructions.

Adaptivity also plays a central role in the study of lower bounds. A common strategy for establishing these bounds is to first address the easier case of non-adaptive RLDCs. Then, the lower bound for adaptive RLDCs is derived by using a generic reduction from adaptive decoders to non-adaptive ones. Alas, known reductions cause an exponential blow-up in the query complexity (or a similar blow-up in the soundness error), resulting in worse lower bounds for adaptive RLDCs [13, 10].

A main question in the study of probabilistic algorithms concerns the strength of two-sided error algorithms vs. one-sided ones. We say that an algorithm has a one-sided error if it never errs on "YES" instances, and a two-sided error if it is allowed to err on "both sides". A major open problem in computational complexity is whether $\mathcal{BPP} = \mathcal{RP}$, which asks whether, in general, an algorithm allowed two-sided error possesses more computational power than one restricted to have a one-sided error. In the standard definition of RLDCs, the decoder can err with a small probability, even when its input is a valid codeword. That is, it is allowed to have two-sided error. In this work, we ask the equivalent $\mathcal{BPP} = \mathcal{RP}$ question for RLCCs: Are one-sided error relaxed local decoders weaker than two-sided error ones? Can we transform any two-sided error decoder, eliminate its errors on valid codewords, to become a one-sided error decoder?

1.1 Our results

Our main result is that for linear codes, two-sided error and adaptivity do not give any strength to RLDCs and RLCCs.

We show a reduction that starts with a relaxed local decoder (resp., corrector) that might query adaptively and err on valid codewords, and ends with a relaxed local decoder (resp., corrector) for the same code, that is non-adaptive and never errs or rejects on valid codewords. The reduction adds at most one additional query. The new soundness error is the sum of the completeness error and soundness error of the original algorithm. Hence the gap between completeness and soundness stays the same. In particular, the error probability, which is the probability to err on *any* specific input (i.e, $\max(\epsilon_{\text{completeness}}, \epsilon_{\text{soundness}})$) is at most doubled.

▶ **Theorem 1.** *Let C be a* linear *systematic*[2] *code.*

If C has a relaxed local decoder of radius ρ with completeness error $\epsilon_{completeness}$, soundness error $\epsilon_{soundness}$ and query complexity q, then it has a one-sided error, non-adaptive *decoder of radius ρ, with soundness error $\epsilon_{completeness} + \epsilon_{soundness}$ and query complexity $q + 1$.*

▶ **Theorem 2.** *Let C be a* linear *code.*

If C has a relaxed local corrector of radius ρ with completeness error $\epsilon_{completeness}$, soundness error $\epsilon_{soundness}$ and query complexity q, then it has a one-sided error, non-adaptive *corrector of radius ρ, with soundness error $\epsilon_{completeness} + \epsilon_{soundness}$ and query complexity $q + 1$.*

[2] A code is called *systematic* if the entire original message is embedded as the first bits of its encoding.

Improved lower bound

Building upon the work of Gur and Lachish [13], Goldreich [10] established lower bounds on the achievable rate by an RLDC with a constant query complexity q. Specifically, the lower bounds are $n \geq k^{1+\Omega(1/q^2)}$ for the non-adaptive case and $n \geq k^{1+\Omega(1/q^3)}$ for adaptive RLDCs. By utilizing Theorem 1, we can remove the adaptivity restriction from the tighter lower bound (for linear codes), leading to the following result:

▶ **Theorem 3.** *Let C be a linear RLDC with message length k, block length n, constant query complexity q and constant correction radius and error probability.*
Then $n \geq k^{1+\Omega(1/q^2)}$.

This represents an improvement over the current state-of-the-art lower bound by Dall'Agnol, Gur and Lachish [9], which is $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$ (although it holds for non-linear codes as well).

Linearity

Our reduction only works for linear codes. Nevertheless, linear codes are an important type of error-correcting codes and have been extensively studied. Virtually all known RLDCs and RLCCs (and their non-relaxed counterparts) constructions are of linear codes.[3]

Non-systematic codes

In Theorem 1, it is assumed that the code is in a systematic form. This is a technical detail, not an inherent limitation. Any RLDC can be transformed to be systematic by adding the message bits to the beginning of its encoding. This transformation results in, at most, a doubling of the block length of the code and a corresponding reduction in the decoding radius by the same factor. Consequently, if the code's rate and decoding radius were initially constant, they remain unchanged after this transformation. In addition, this transformation enables the elimination of the systematic requirement in Theorem 3.

It is worth noting that any linear code can be made systematic through a basis change without altering its block length. While such a transformation does not affect the set of valid codewords, it does alter the encoding function. For RLDCs, this implies that the code's decoder may no longer be valid after the transformation.

Known reductions

We note two well-known immediate reductions from adaptive to non-adaptive local algorithms. These reductions date back to [17], which stated them for non-relaxed LDCs, but they also apply to relaxed ones.

The first reduction is to replace each of the q adaptive queries with multiple non-adaptive ones. We replace the i-th query with 2^{i-1} queries, one for each possible result of the previous queries. This reduction yields an exponential blowup in the query complexity. The second reduction is to "guess" the result of the first $q-1$ queries, and query a set of indices based on that guess. This reduction exponentially decreases the algorithm's soundness.

[3] A remarkable exception is multiplicity codes, which are not linear. Fortunately, their codewords constitute a subgroup (the sum of two codewords is a codeword), so our framework still covers them. See Definition 4.

In light of the above, even a reduction with a polynomial blowup in the query complexity would have been an exciting result.

Our contribution

Ben-Sasson, Harsh and, Raskhodnikov [4] showed that for testing *linear* properties, adaptivity and two-sided error do not help. Our work is based on [4], and extends their reductions to the setting of RLDCs and RLCCs.

One technical difficulty in extending their work is that decoders and correctors get an index as an input, in addition to the noisy codeword. This difficulty turns out to be minor. The major difficulty is regarding the actual task at hand. First, testers work under the promise that the input either has the property, or is *far* from any element having it. In contrast, decoders and correctors work under the promise that the input is *close* to the code (or in it). Furthermore, the "output type" is different between those algorithms. On the one hand, a tester has a binary verdict - it accepts or rejects the input. On the other hand, the output of decoders and correctors is not binary - it is a symbol of the message, or a rejection symbol.

To overcome those differences, we introduce the abstract notion of *additive promise problem* (Definition 4). We show that testing linear properties and relaxed decoding/correction of linear codes satisfy this notion.

We then present a reduction that applies to *any* promise problem satisfying this notion. The reduction is a generalization of the one in [4]. The generalization is done by decoupling the logic of the reduction from the testing context. We prove the main results by combining the interpretation of relaxed decoding/correction as promise problems with the generic reduction. Arguably, the generalized reduction also provides a cleaner and more straightforward presentation of [4].

We consider the main contribution of this work to be conceptual. By formalizing the notion of additive promise problem, we demonstrate what is the minimal set of requirements necessary for the techniques of [4] to be applicable. We believe the new notion is natural and intuitive, and hope it will find more applications in the future.

1.2 Motivation

The primary motivation for this work is to enhance our understanding of RLDCs and RLCCs. Additionally, we outline specific motivations as follows.

Applications for lower bounds

Our results show that, for linear codes, any lower bound for non-adaptive, one-sided error RLDCs (resp., RLCCs) can be translated to a lower bound for adaptive, two-sided error RLDCs (resp., RLCCs). Applying this outcome to the work of [10] yields Theorem 3, which is the current best lower bound for linear, adaptive, two-sided error RLDCs. However, this bound is not known to be tight; specifically, the best RLCC construction with constant query complexity q, of Asadi and Shinkar [2], achieves block length $n = k^{1+O(1/q)}$. As our reduction is applicable to any RLCC, it holds potential for accommodating future improvements in lower bounds

In addition, in the case of linear codes, any lower bound applicable to RLDCs also serves as a lower bound for RLCCs, but the reverse is not true. This arises from the fact that every linear code can be represented systematically, with the initial bits of the codeword corresponding to the original message. Hence, for such codes, the corrector can be used as a decoder, so RLCCs are stronger objects than RLDCs. Our reduction operates independently on each type, implying its potential application for improved lower bounds on RLCCs, which may surpass those for RLDCs.

We note that the work of [9] extends the result of [13] to adaptive, two-sided error RLDCs, for all codes, including non-linear ones. However, this extension does not rely on a generic reduction and is notably highly involved. In contrast, our reduction is arguably much simpler, offering an alternative, simpler proof for linear codes.

Constructions

Virtually all known constructions of RLDCs and RLCCs are non-adaptive, linear, and have one-sided error. Our results show that this should not be a surprise. It is impossible to use adaptivity or two-sided error to improve constructions of linear RLDCs / RLCCs.

Definitions

In some works (e.g, [6, 15, 2, 7]), the definition of RLDC requires it to have one-sided error (i.e., it is not an additional property that a decoder might have). Other works (e.g., [3, 9]) use the same definition we gave above. Our result settles this nuance in the definitions - both are equivalent (for linear codes).

1.3 Technical overview

We next give a high-level sketch of the reduction.

Promise problems

We prove Theorem 1 and Theorem 2 by proving a general result on a family of promise problems. First, a promise problem is a pair of disjoint sets, Y (the YES-instances) and N (the NO-instances). A randomized algorithm for a promise problem gets as input $x \in Y \cup N$ (we sometimes call $Y \cup N$ "the promise") and outputs YES or NO. If $x \in Y$ then the algorithm must output YES with high probability. Similarly, if $x \in N$ it must output NO with high probability.

The main new idea we introduce in this work is of *additive* promise problems.

▶ **Definition 4.** *A promise problem $(Y, N) \subseteq \{0,1\}^n$ is* additive *if it satisfies the following conditions:*

1. *(YES-instances are a linear subspace[4]) For every $x, y \in Y$, $x + y \in Y$*
2. *(NO-instances are a collection of cosets) For every $x \in N, y \in Y$, $x + y \in N$*

This definition can be generalized to any abelian group instead of $\{0,1\}^n$. For simplicity, in this work we focus on $\{0,1\}^n$.

Testing linear properties

As a demonstration for the new definition, we next show that property testing, when the tested property $\Pi \subseteq \{0,1\}^n$ is a linear subspace, is an additive promise problem. The YES-instances in this case are the elements of the tested property. That is, $Y = \Pi$. The NO-instances are the elements ϵ-far from every YES-instances. Namely,

[4] Strictly speaking, the requirement is that the YES-instances are a *subgroup*. For $\{0,1\}^n$ these requirements are equivalent.

$$N = \{x \in \{0,1\}^n \mid \forall y \in Y,\ \mathrm{dist}(x,y) > \epsilon\}$$

where $\mathrm{dist}(x,y)$ is the relative Hamming distance between x and y, (i.e., $\mathrm{dist}(x,y) = \frac{|\{x_i \neq y_i | i \in [n]\}|}{n}$).

The first item of Definition 4 follows from the assumptions that the property Π is linear. For the second item, let $x \in N, y \in Y$. We need to show that $x + y \in N$. I.e, that $x + y$ is ϵ-far from every $y' \in Y$. Indeed, for every $y' \in Y$ we have

$$\mathrm{dist}(x+y, y') = \mathrm{dist}(x, y' - y) > \epsilon.$$

The equality holds because in general, $\mathrm{dist}(a,b) = \mathrm{dist}(a+c, b+c)$ for every a, b, c. The inequality holds because, since y and y' are in the linear space Y then $y' - y \in Y$, and since $x \in N$ it is ϵ-far from every element in Y.

In Section 3, we show how to interpret relaxed decoding and correction of linear codes as promise problems. This step might result in performing one additional query. We use similar arguments as in the proof above to show that the resulting promise problems are additive.

The reduction

Next, we construct a reduction from adaptive, two-sided error local algorithms to non-adaptive, one-sided error ones that works for *any* additive promise problem.

▶ **Theorem 5.** *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an adaptive algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has a non-adaptive, one-sided error algorithm A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

By applying Theorem 5 to relaxed decoding we prove Theorem 1, and by applying it to relaxed correction we prove Theorem 2.

The reduction works in two steps. The first step ensures that the algorithm never errs on YES-instances. We start with an adaptive, two-sided error arbitrary algorithm, and transform it to have one-sided error (and it remains adaptive). This step does not increase the query complexity. If the original algorithm errs on YES-instances with probability at most ϵ_Y and on NO-instances with probability at most ϵ_N, then the transformed algorithm errs on NO-instances with probability at most $\epsilon_Y + \epsilon_N$.

The second step handles adaptivity. We start with an adaptive, one-sided error algorithm, and transform it to a non-adaptive algorithm (that still has one-sided error). This step maintains the query complexity and the soundness error.

We next describe the two reductions.

Two-sided to one-sided error

Every randomized algorithm A can be described as a distribution over a set of deterministic decision trees. Each leaf of each decision tree is labeled with YES or NO, which is the output of the algorithm when that tree is chosen. The first step of the reduction is to *relabel* the leaves of all trees, in the following way: If there is an input $x \in Y$ that "leads" to this leaf, then it is relabeled to YES. This step is necessary to get a one-sided error algorithm. However, this transformation may not maintain the algorithm's soundness. In Section 4.1, we explain the issue in detail.

The solution is to modify the algorithm. Instead of using the (relabeled) decision trees of A with the given input x, choose a random YES-instance y, and use the tree as if the input was $x+y$. Since (Y,N) is an additive promise problem, if $x \in Y$ then $x+y \in Y$ for any (randomly chosen) y, and the original algorithm's completeness can be used. Similarly, if $x \in N$ then $x+y \in N$ for any y, and the soundness of the original algorithm can be used. In Lemma 16, we prove that with this modification the transformation maintains the sum of soundness and completeness error.

Adaptivity

Consider an adaptive, one-sided error algorithm A. Without loss of generality, the only freedom A has is in choosing its queries. Once it queried an input x, it must output YES if there exists a YES-instance consistent with the queries. Otherwise, when no YES-instance is consistent, then x cannot be a YES-instance and w.l.o.g A outputs NO.

The new non-adaptive algorithm A' works as follows: On input x, choose a random YES-instance y. Query x on all indices A would have queried y, and output YES if the partial view of x is consistent with some YES-instance (which might be different than y).

The new algorithm is non-adaptive since now it determines its queries independently of its input. Its query complexity is maintained, and it has one-sided error (as it always outputs YES for YES-instances). In Lemma 20, we show that its soundness error is also maintained. This is done by relating the probability A' outputs YES on some specific x to the *average* probability A outputs YES for a random element of the set $x+Y$.

1.4 Related work

Error correcting codes date back to the seminal works of Shannon [23] and Hamming [16]. LDCs, LCCs and their relaxed counterparts have attracted significant attention in recent years. See the works of Yekhanin [26] and Kopparty and Saraf [19] and references within for comprehensive surveys of LDCs, LCCs and their applications.

RLDCs and RLCCs constructions

The constructions of RLDCs and RLCCs can be separated into two regimes of parameters: constant query complexity, and constant rate. In the constant rate regime, the state-of-the-art code is the construction of Cohen and Yankovitz [8]. This construction is of a linear RLCC with rate arbitrarily close to 1, and query complexity $q = (\log n)^{2+o(1)}$. This construction builds upon the result of Kumar and Mon [20], which shows a similar code but with query complexity $q = (\log n)^{O(1)}$.

In the constant query regime, the original work of [3] achieves RLDC with constant query complexity $O(q)$ and block length $n = O(k^{1+1/\sqrt{q}})$. The work of [15] introduced the notion of RLCCs, and constructed such a code with constant query complexity, but with worse block length. Chiesa, Gur, and Shinkar [6] constructed an improved RLCC, matching the block length of [3].

The current state-of-the-art construction is of Asadi and Shinkar [2], which builds upon [15] and [6]. Their construction is of RLCC and RLDC with constant query complexity $O(q)$ and block length $n = O(k^{1+1/q})$. This is the first construction of RLDC, in the constant query complexity regime, with better block length than of [3].

We remark that all the above constructions are linear, non-adaptive, and have one-sided error.

Lower bounds

In recent decades, extensive research has been conducted on lower bounds for (non-relaxed) LDCs in various regimes [17, 18, 24, 25, 1]. Gur and Lachish [13] presented the first lower bound for relaxed LDCs. Such lower bounds are arguably harder to obtain, as RLDCs are weaker objects than LDCs. Specifically, they showed that any non-adaptive RLDC requires a block length of $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$. For the adaptive case, they established a lower bound of $n \geq k^{1+\Omega(\frac{1}{2^{2q} \log^2 q})}$, by using the known reduction mentioned above, that causes an exponential blowup in the query complexity.

The result of [13] was extended to additional settings, such as proofs of proximity, property testing and to adaptive settings by Dall'Agnol, Gur and Lachish [9]. Their result extends the lower bound of $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$ to *adaptive* RLDCs.

Goldreich [10] surveyed and simplified the work of [13], without employing the new techniques of [9]). He established an improved bound of $n \geq k^{1+\Omega(1/q^2)}$ for the non-adaptive case, and a bound of $n \geq k^{1+\Omega(1/q^3)}$ for the adaptive case (which is weaker than the one presented in [9]).

Locally testable codes and RLDCs

Our work follows the theme of extending ideas from the area of *LTCs* (locally testable codes [12]) to RLDCs. Refer to the full paper for an analysis comparing RLDCs and LTCs, including details about their similarities, distinctions, and relevance to this study.

1.5 Open problems

We conclude this section with a few questions we leave for future research.

Non-linear codes

Our result holds only for linear codes. A natural open problem is to extend the result to non-linear codes, or to show that for non-linear codes, adaptivity and / or two-sided error *do* give more power. Our reduction heavily relies on linearity, so we believe other techniques than the ones used in this paper will be needed.

Adaptivity of LDCs and LCCs

The power of adaptivity for (non-relaxed) LDCs and LCCs is an interesting open problem. As we discuss in Section 3, our framework of additive problems does not seem to cover them. This leaves the question open, even for linear codes. To the best of our knowledge, there are currently no known reductions from adaptive to non-adaptive LDCs, besides the ones described above. Even a reduction that polynomially increases the query complexity, or slightly increases the code's block length, would be an interesting result.

A main open problem is to separate the power of LDCs and that of relaxed LDCs. Constructing an adaptive (linear) LDC might aid in this effort, showing that LDCs and relaxed LDC differ regarding adaptivity.

Efficient reduction

The reduction we propose could potentially incur an exponential cost in terms of time complexity. Addressing the challenge of making the reduction efficient, (i.e., ensuring that the transformed algorithm runs in polynomial time), remains an open problem. See Remark 19 for details.

2 Definitions and preliminaries

Refer to the full paper for basic notations used herein.

2.1 Error correcting codes

Throughout, an error correcting code C with *message length* k and *block length* n is a function $C : \{0,1\}^k \to \{0,1\}^n$. For simplicity, we consider only binary alphabet in this work. We identify a code with its image, i.e, $C \subseteq \{0,1\}^n$.

A code C is *linear* if it is a linear function (or, equivalently, if C as a set is closed under addition).

▶ **Definition 6.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be an error correcting code. A relaxed* decoder *of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [k]$, outputs an element of $\{0,1,\perp\}$, and satisfies the following two requirements:*

1. *(completeness) If $x = C(y)$ for some $y \in \{0,1\}^k$ then $A^x(i) = y_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(relaxed local decoding) If there exists $y \in \{0,1\}^k$ such that $dist(x, C(y)) < \rho$, then $A^x(i) \in \{y_i, \perp\}$ with probability at least $1 - \epsilon_{soundness}$.*

The probabilities are over the internal randomness of A.

▶ **Definition 7.** *Let $C \subseteq \{0,1\}^n$ be an error correcting code. A relaxed* corrector *of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [n]$, outputs an element of $\{0,1,\perp\}$, and satisfies the following two requirements:*

1. *(completeness) If $x \in C$ then $A^x(i) = x_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(relaxed local correction) If there exists $c \in C$ such that $dist(x, c) < \rho$, then $A^x(i) \in \{c_i, \perp\}$ with probability at least $1 - \epsilon_{soundness}$.*

The probabilities are over the internal randomness of A.

In both definitions we call $\epsilon_{\text{completeness}}$ the *completeness error*, and $\epsilon_{\text{soundness}}$ the *soundness error*.

In what follows we use the term *local algorithm* to refer to an algorithm which is a relaxed local decoder or a relaxed local corrector. A local algorithm has *one-sided error* if its completeness error is 0 (i.e., if it never errs on valid codewords). We say that a local algorithm has *query complexity* $q = q(n)$ if, on input i, and with oracle access to any $x \in \{0,1\}^n$, the corrector makes at most $q(n)$ queries. We say that a local algorithm is *non-adaptive* if it determines all its queries based on its explicit input (namely, the index to decode / correct) and internal coin tosses, independently of the specific x to which it is given oracle access. Otherwise, we say that it is *adaptive*.

For an RLDC, we view its decoder A of a code as a set of k decoders $A_1, \ldots, A_k$, where $A_i(x) = A^x(i)$. We call A_i the *decoders* of C. Similarly, we view the corrector of a RLCC as a set of n correctors, $A_1, \ldots, A_n$, and call them the *correctors* of C. The benefit of this view is that each A_i is now an algorithm that gets a single (implicit) input $x \in \{0,1\}^n$.

2.2 Promise problems

▶ **Definition 8.** *A* Promise Problem *is couple $(Y, N) \subseteq \{0,1\}^n$ such that $Y \cap N = \emptyset$. We call Y the* YES-instances *of the problem, and N the* NO-instances *of the problem.*

▶ **Definition 9.** *An* algorithm for a promise problem $(Y, N) \subseteq \{0,1\}^n$ *with* completeness error $\epsilon_Y > 0$ *and* soundness error $\epsilon_N > 0$ *is a randomized procedure that gets oracle access to an input* $x \in \{0,1\}^n$*, outputs YES or NO, and satisfies the following conditions:*

1. *(completeness) If* $x \in Y$ *then it outputs YES with probability at least* $1 - \epsilon_Y$*.*
2. *(soundness) If* $x \in N$ *then its outputs NO with probability at least* $1 - \epsilon_N$*.*

We define query complexity and adaptivity of promise problem algorithms as they are defined in Definition 7. We say that a promise problem algorithm has *one-sided* error if its completeness error is 0 (i.e., if it never errs on YES-instances).

The main new definition of this work is of additive promise problems. See Definition 4.

3 Relaxed decoding and correction as additive promise problems

In this section, we show how to interpret relaxed decoding and relaxed correction of linear codes as promise problems. We do that in Section 3.1. We then show, in Section 3.2, that the resulting promise problems are additive.

3.1 Interpretation as promise problems

We start by showing how to interpret relaxed correction as a promise problem. There are three possible values for the output of a relaxed local corrector: $0, 1$ or $\perp$.[5] In contrast, an algorithm for a promise problem has only two possible outputs: YES and NO. We must specify how to translate a correction problem (with multiple possible output values) into a yes/no question. The following observation enables us to do that.

▷ Claim 10. If a code has a corrector A, then it has a corrector A' such that for every $x \in \{0,1\}^n$, the output of A' for index i is x_i or $\perp$. A' has the same completeness and soundness errors as A, and it might make one additional query.

Proof. The new corrector A'_i works according to the following rule:

$$A'_i(x) = \begin{cases} A_i(x), & \text{if } A_i(x) \in \{x_i, \perp\} \\ \perp, & \text{otherwise.} \end{cases}$$

From the construction, the output of $A'_i(x)$ is x_i or $\perp$ for every input x (and never $1 - x_i$). A' makes at most one additional query compared to A in order to retrieve the value x_i.

We next show that A'_i satisfies the required completeness and soundness (Definition 7). For completeness, if $x \in C$ then $A_i(x) = x_i$ with probability $1 - \epsilon_{\text{completeness}}$, and hence $A'_i(x) = x_i$ with the same probability.

For soundness, assume there exists $c \in C$ such that $\text{dist}(x, c) < \rho$. We need to show $A'_i(x) \in \{c_i, \perp\}$ with high probability. Consider the case $c_i = x_i$. From the soundness of A_i, $A_i(x) \in \{x_i, \perp\}$ with probability at least $1 - \epsilon_{\text{soundness}}$, and hence $A'_i(x) \in \{x_i, \perp\} = \{c_i, \perp\}$ with the same probability.

Otherwise, $c_i \neq x_i$. With probability at least $1 - \epsilon_{\text{soundness}}$, the output of A_i is c_i or $\perp$. From the construction, whenever the output of A_i is c_i or $\perp$, the output of A'_i is $\perp$. Hence, with the same probability, $A'_i(x) = \perp \in \{c_i, \perp\}$. ◁

By using Claim 10, we can replace item 2 of Definition 7, and assume the corrector always outputs x_i or $\perp$:

[5] For a larger alphabet, the number of possible outputs of a corrector is the size of the alphabet $+1$.

▶ **Definition 11** (alternative definition of RLCCs). *A relaxed corrector of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [n]$, outputs x_i or $\perp$, and satisfies the following:*

1. *(completeness) If $x \in C$ then $A^x(i) = x_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(soundness) If there exists $c \in C$ such that $dist(x, c) < \rho$ and $x_i \neq c_i$, then $A_i(x) = \perp$ with probability at least $1 - \epsilon_{soundness}$.*

Definition 11 allows us to treat a corrector as having a "binary" output; On input x, the output of A_i is either $x_i \in \{0,1\}$ (signifying "accept") or $\perp$ (signifying "reject"). We can now phrase relaxed correction as a promise problem, as follows[6]:

▶ **Definition 12.** *Let $C \subset \{0,1\}^n$ be an error correcting code, let $\rho > 0$ and let $i \in [n]$. The* promise problem of relaxed correction *of C at index i with correction radius ρ is defined by:*

1. *(YES-instances are the codewords)*
   $Y = C$
2. *(NO-instances are the inputs a corrector rejects)*
   $N = \{x \in \{0,1\}^n \mid \exists c \in C \text{ with } dist(x,c) < \rho \text{ and } x_i \neq c_i\}$.

The promise problem of relaxed correction is equivalent to relaxed correction of codes (of Definition 11) in the following sense:

▷ Claim 13. A corrector for index i of the code C can be translated to an algorithm for the promise problem of relaxed correction of C at index i (Definition 12) by identifying the outputs $0, 1$ as "YES" and the output $\perp$ as "NO". The parameters $\epsilon_{\text{completeness}}, \epsilon_{\text{soundness}}, \rho$ remain the same.

Relaxed decoding

From the hypothesis for RLDCs, the code at hand is in systematic form. That is, we assume that the first k bits of each codeword are the message encoded in it. Hence, a decoder is simply a corrector that needs to "correct" only the first k bits of the input. The observation above (Claim 10) holds for relaxed local decoders as well. Hence, we can assume w.l.o.g that the output of the decoder, for input x and index $i \in [k]$, is either x_i or $\perp$. This allows us to handle RLDCs in a manner similar to RLCCs. For further details, please refer to the full paper."

3.2 Additive promise problems

In this section, we show that the promise problems formulated above for relaxed decoding and relaxed correction, have the special property of being *additive* (Definition 4).

▷ Claim 14. The relaxed correction and relaxed decoding promise problems for linear codes is additive.

We prove the claim for relaxed correction. The proof for relaxed decoding is the same, with the restriction that $i \in [k]$ (instead of in $[n]$).

---

[6] Another possible formulation is $Y = \{(x,b) \mid x \in C \text{ and } b = x_i\}$ and $N = \{(x,b) \mid \exists c \in C \text{ with } \mathrm{dist}(x,c) < \rho \text{ and } c_i \neq b\}$. This formulation preserves better the decoding "flavor" of the problem. We use the formulation of Definition 11, as it emphasizes the similarity to testing, with $\perp$ corresponding to "reject" and any other output corresponding to "accept".

Proof. Let C be a *linear* error correcting code, let $\rho > 0$ and let $i \in [n]$. Let Y, N be as in Definition 12.

From the linearity assumption $Y = C$ is a linear subspace of $\{0,1\}^n$, and the first item of Definition 4 holds.

To show the second item, let $x \in N, y \in Y$. We need to show that $x + y \in N$. Since $x \in N$, there exists a codeword $c \in C$ such that $\mathrm{dist}(x, c) < \rho$ and $x_i \neq c_i$. Define $c' = c + y$. c' is a codeword in C, since it is a sum of two codewords, and C is a linear code. We get $\mathrm{dist}(x + y, c') = \mathrm{dist}(x + y, c + y) = \mathrm{dist}(x, c) < \rho$, and $(x + y)_i = x_i + y_i \neq c_i + y_i = c'_i$. Hence $x + y \in N$. ◁

Non-relaxed LDCs

We remark that *non-relaxed* decoding / correction (for a specific index) does not seem to fit into our new framework. Refer to the full paper for an in-depth discussion on this topic.

4 The reduction

In this section, we prove our main results, Theorem 1 and Theorem 2.

We prove these theorems by constructing the following, more general reduction:

▶ **Theorem 15** (Restatement of Theorem 5). *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an adaptive algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has a* one-sided error, non-adaptive algorithm *A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

Theorem 1 and Theorem 2 are direct corollary of Theorem 15 applied to the relaxed decoding and correcting promise problem.

The proof of Theorem 15 has two steps. The first is a reduction from two-sided error to one-sided error algorithms. We show this reduction in Section 4.1. The second step is a reduction from one-sided, adaptive algorithms to one-sided, non-adaptive algorithms. We show this reduction in Section 4.2.

4.1 From two-sided to one-sided error

In this section, we show a reduction from two-sided error algorithms to one-sided error ones for additive promise problems. The reduction does not change the query complexity of the algorithm, and maintains the sum of the completeness and soundness errors.

▶ **Lemma 16.** *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an (adaptive) algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has an (adaptive) one-sided error algorithm A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

Randomized algorithms as distributions over decision trees

Consider some randomized algorithm A for a promise problem. A can be described as a distribution D_A over a set of deterministic decision trees $\Upsilon_A = \{\Gamma_1, \Gamma_2, \ldots\}$. We denote by $\Gamma \sim D_A$ a tree chosen randomly from Υ_A according to the distribution D_A. Each leaf ℓ of each tree corresponds to a set of indices $I = (i_1, \ldots, i_t) \in [n]^t$ that are queried along the path leading to ℓ, and the corresponding values $\sigma = (\sigma_1, \ldots, \sigma_t) \in \{0,1\}^t$ at these indices. We identify each leaf with the corresponding indices and values and write $\ell = (I, \sigma)$.

For an input x, we denote by $\Gamma(x)$ the *leaf* of Γ at the end of the path corresponding to querying x. That is, $\Gamma(x) = \ell$ if $x|_I = \sigma$, where $\ell = (I, \sigma)$, and $x|_I$ is the restriction of x to indices I. Each leaf is labeled with YES or NO.

We can now describe the operation of A on input x as follows: It chooses a $\Gamma \sim D_A$ and outputs the label of $\Gamma(x)$.

Relabeling decision trees

To convert an algorithm A to have one-sided error, we first go over all of $\Gamma \in \Upsilon_A$ and relabel them, so each Γ will have one-sided error. The relabeling works as follows: For every leaf ℓ of Γ, if there exists a YES-instance y such that $\Gamma(y) = \ell$, relabel ℓ with YES. We denote the relabeled tree by Γ_c and call it the *one-sided error relabeling* of Γ.

This step is necessary to get a one-sided error algorithm; as long as there exists $y \in Y$ and Γ such that the label of $\Gamma(y)$ is NO, there is some probability that A outputs NO for a YES-instance. The issue, however, is that this transformation may not maintain the algorithm's soundness. There might be NO-instances that the new algorithm (wrongly) accepts with a high probability.

The soundness issue with relabeling

Pretend the transformed algorithm A' worked as follows. For an input x, choose a random decision tree $\Gamma \sim D_A$, relabel the tree to get Γ_c, and output the label of $\Gamma_c(x)$ (instead of using $\Gamma(x)$ as the original algorithm did). This new algorithm has one-sided error. In fact, we can transform *any* algorithm this way to have one-sided error, so we should not expect it to maintain soundness.

To see that the soundness is not necessarily maintained, consider some leaf ℓ of a tree Γ that was relabeled from NO to YES. Let $x \in \{0,1\}^n$ such that $\Gamma(x) = \ell$. If $x \in Y$, the relabeling was beneficial. Before the relabeling, the algorithm returned a wrong output for x (whenever it used the tree Γ), and now it returns the correct output. However, what if x is a NO-instance? In this case, after the relabeling we return the *wrong* output for x each time Γ is used. The tree Γ is sampled according to the distribution D_A, which is arbitrary. If D_A gives much weight to Γ (say, it is chosen with probability $\frac{1}{2}$), then the new algorithm returns the wrong output for x with high probability.[7] This implies that A' does not have the required soundness property, as it should hold for *every* NO-instance.

The actual reduction

The solution is to modify the algorithm in the following way. Instead of deterministically returning the label of $\Gamma_c(x)$ (after Γ was chosen at random), the transformed algorithm outputs the label of $\Gamma_c(x + y)$ for a random $y \in Y$. We give a formal description of the transformation in Algorithm 1. Now, even if $\Gamma_c(x)$ was relabeled to YES, we output its label with a small probability. In the proof of Lemma 16, we show that the soundness error of the transformed algorithm is at most $\epsilon_Y + \epsilon_N$.[8]

---

[7] Even if there is no one heavy tree, relabels of many leaves in different trees might have the same effect if their total weight is high.

[8] Recall that ϵ_N is error probability of the original algorithm for NO-instances, and ϵ_Y is its error probability for YES-instances.

Algorithm 1 One-sided error local algorithm, A'.

Input: Oracle access to $x \in \{0,1\}^n$.
Output: YES or NO.

1. Choose $\Gamma \sim D_A$.
2. Choose $y \in Y$ uniformly at random.
3. Output the label of $\Gamma_c(x+y)$.

Query complexity

The relabeling does not change the query complexity (i.e., depth) of the decision trees. Hence, the query complexity of A' is the same as that of A.

One-sided error

Let $x \in Y$. We argue that A' always outputs YES for x. From the definition of A', its output for x is the label of $\Gamma_c(x+y)$ for some $y \in Y$. Since (Y, N) is an additive problem and $x \in Y$, we get that $x+y \in Y$. From the relabeling scheme for Γ_c, the label of $\Gamma_c(y')$ is YES for every $y' \in Y$, and in particular for $y' = x+y$. Hence, the output of A' for x is YES.

Before arguing about the soundness of the transformed algorithm, we need the following preparations.

Probability of hitting a specific leaf

We stated above that for an input x, the modified algorithm uses the label of a specific leaf ℓ with a small probability. We next calculate this probability (conditioning on first choosing the tree Γ_c of ℓ). There are $|Y|$ possible options for y. The leaf ℓ is used if the chosen y satisfies $\Gamma_c(x+y) = \ell$. Hence, we choose the label of ℓ with probability $\frac{|(x+Y)\cap\Gamma^{-1}(\ell)|}{|Y|}$, where $x+Y = \{x+y \mid y \in Y\} \subseteq \{0,1\}^n$, and $\Gamma^{-1}(\ell) = \{z \mid \Gamma(z) = \ell\} \subseteq \{0,1\}^n$. We argue that this probability is either 0 (when there is no $y \in Y$ such that $\Gamma(x+y) = \ell$ so $(x+Y)\cap\Gamma^{-1}(\ell)$ is empty), or is equal to a quantity not depending on x.

▶ **Lemma 17.** *Let $Y \subseteq \{0,1\}^n$ be a subspace of $\{0,1\}^n$, and fix a decision tree Γ and a leaf $\ell = (I,\sigma)$. Define $U = \{u \in Y \mid u|_I = 0\}$. Then for every $x \in \{0,1\}^n$, if there exists $y \in Y$ such that $\Gamma(x+y) = \ell$, then $|(x+Y)\cap\Gamma^{-1}(\ell)| = |U|$.*

Proof. First, notice that U is not empty since the all-zeros string is in Y (as Y is a linear subspace). Furthermore, U is a subspace of Y, since if $u, u' \in U$ then $(u+u')|_I = u|_I + u'|_I = 0$ and $u+u' \in U$. We argue that $(x+Y)\cap\Gamma^{-1}(\ell)$ is a coset of U, hence having the same size as U. Namely, we claim:

$$(x+Y)\cap\Gamma^{-1}(\ell) = x+y+U$$

where y is an element of Y such that $\Gamma(x+y) = \ell$.

We begin by proving the inclusion $(x+Y)\cap\Gamma^{-1}(\ell) \subseteq x+y+U$. Let $x+y' \in (x+Y)\cap\Gamma^{-1}(\ell)$. That is, $(x+y')|_I = \sigma$. Define $u = y'-y$. Since $(x+y)|_I = (x+y')|_I = \sigma$, we have $u|_I = ((x+y')-(x+y))|_I = 0$ and $u \in U$. Therefore $x+y' = x+y+u \in x+y+U$.

To prove that $(x+Y)\cap\Gamma^{-1}(\ell) \supseteq x+y+U$, let $u \in U$. Since $y \in Y$ and $u \in U \subseteq Y$, and Y is closed under addition, we have $y+u \in Y$ and $x+y+u \in x+Y$. Next, $(x+y+u)|_I = (x+y)_I + u|_I = (x+y)_I = \sigma$ and hence $x+y+u \in \Gamma^{-1}(\ell)$. ◀

We are now ready to prove Lemma 16.

Proof of Lemma 16. We proved above that the reduction maintains the query complexity of the algorithm, and that the transformed algorithm has one-sided error. We are left with establishing an upper bound for its soundness error.

Let $x \in N$. We need to show that A' outputs NO for x with probability at least $1-\epsilon_{\mathrm{Y}}-\epsilon_{\mathrm{N}}$. That is, we need to prove $\Pr[A'(x) = \text{ YES }] < \epsilon_{\mathrm{Y}} + \epsilon_{\mathrm{N}}$.

The probability for (wrongly) outputting YES for x may increase due to the transformation. Nevertheless, it does not increase too much. We argue the transformation does not decrease the gap between the expected probability of returning YES for a random element of Y and the expected probability of outputting YES for a random element of $x + Y$. That is, we claim:

▷ Claim 18. For every $x \in N$:

$$\begin{aligned}\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \text{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \text{YES}]] \\ \leq \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(y) = \text{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(x+y) = \text{YES}]]\end{aligned}$$

where the probabilities are over the internal randomness of A and A'.

Proof of Claim 18. To prove this claim, it is enough to show that relabeling one leaf ℓ of one decision tree does not decrease the gap. Then we obtain the claim by relabeling one leaf at a time, to get A' from A.

Assume ℓ was relabeled from NO to YES. Let $G := Y \cap \Gamma^{-1}(\ell)$ be the strings $y \in Y$ such that $\Gamma(y) = \ell$ (these are the "Good" strings, which are now labeled YES and are in Y). The set G is not empty since we relabel ℓ only if there exists $y \in Y$ such that $\Gamma(y) = \ell$, i.e. $y \in G$. Let $B := (x + Y) \cap \Gamma^{-1}(\ell)$ be the strings $x + y \in x + Y$ such that $\Gamma(x + y) = \ell$ (these are the "Bad" strings, which are now labeled YES but in $x + Y \subseteq N$).

Every string in $G \cup B$ was rejected before the relabeling but is now accepted. The algorithm's behavior on the other elements in Y and $x+Y$ is unaltered. Hence, $\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \text{YES}]]$ increases by $D(\Gamma) \cdot \frac{|G|}{|Y|}$ when ℓ is relabeled (recall that $D(\Gamma)$ is the probability the algorithm chooses Γ). Similarly, $\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x + y) = \text{YES}]]$ increases by $D(\Gamma) \cdot \frac{|B|}{|Y|}$. It suffices to show that $|G| \geq |B|$. Intuitively, this means any "harm" done to x by the relabeling (an element of B that increases the probability to wrongly output YES) is "compensated" by the relabeling (by an element of G improving the algorithm's completeness).

If B is empty, we are done. Otherwise, due to Lemma 17, $|B| = |U|$ and $|G| = |U|$, and we conclude that $|G| = |B|$. ◁

From the claim, the soundness error for x gets worse by an amount bounded by the completeness improvement. Since the completeness error reduces from ϵ_{Y} to 0, the soundness error for x increases by at most ϵ_{Y}.

More formally, from Claim 18, the prefect completeness of A' and the completeness error of A:

$$\begin{aligned}&\mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(x+y) = \text{YES}]] \\ &\leq \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(y) = \text{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \text{YES}]] + \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \text{YES}]] \qquad (1)\\ &\leq 1 - (1-\epsilon_{\mathrm{Y}}) + \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \text{YES}]] = \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \text{YES}]] + \epsilon_{\mathrm{Y}}.\end{aligned}$$

Now, $x + y \in N$ for every $y \in Y$, since $x \in N$ and (Y, N) is additive. Hence, applying the soundness of A to *every* $x + y$, we have $\mathbb{E}_{y \in Y}[\Pr[A(x + y) = \text{ YES }]] < \epsilon_{\mathrm{N}}$.

In addition, from the definition of A', $\Pr[A'(x + y) = \text{YES}] = \Pr[A'(x) = \text{YES}]$ for every $y \in Y$. Hence, from equation 1 we get that $\Pr[A'(x) = \text{YES}] < \epsilon_{\mathrm{Y}} + \epsilon_{\mathrm{N}}$. ◀

▶ Remark 19. As described, the reduction might be expansive in the terms of time complexity. Sampling a YES instance uniformly at random and relabeling the trees can be computationally intensive tasks.

On the positive side, sampling a YES instance can be implemented in polynomial time. The set of YES instances Y forms a subspace, allowing for efficient sampling by taking a random linear combination of elements in a basis of Y. In the context of codes, this is equivalent to encoding a random message using a generating matrix of the code, where the columns of the matrix constitute a basis for the code.

Unfortunately, the relabeling process, even for a single leaf, appears to be computationally hard. Brute-force relabeling involves iterating over a potentially exponentially large set of candidates (all elements of $\{0, 1\}^n$ consistent with the leaf) and checking their membership in Y. Checking membership in Y can be preformed efficiently by considering the dual space of Y. However, we do not see a way to eliminate the iteration over an exponential set of candidates, and leave this question for further research.

4.2 From adaptive to non-adaptive algorithms

In this section, we show a reduction from one-sided error adaptive to (one-sided error) non-adaptive algorithms for additive promise problems. The reduction maintains the algorithm's query complexity and its soundness error.

▶ **Lemma 20.** *Let $(Y, N) \subseteq \{0, 1\}^n$ be an additive promise problem. If (Y, N) has an* one-sided error, adaptive algorithm *A with soundness error ϵ_N and query complexity q, it has an one-sided error* non-adaptive algorithm *A' with soundness error ϵ_N and query complexity q.*

Proof. Let $D = D_A$ be a distribution over decision trees $\Upsilon = \Upsilon_A$ corresponding to A.

We first observe that we can assume w.l.o.g that the label of each leaf $\ell = (I, \sigma)$ is YES if and only if $\exists y \in Y$ such that $y|_I = \sigma$. Since A never errs on YES-instances, if there exists $y \in Y$ such that $y|_I = \sigma$, then ℓ must be labeled YES. On the other hand, we can assume that if no such $y \in Y$ exists, then ℓ is labeled NO. Otherwise, ℓ can be relabeled from YES to NO while only improving the algorithm's soundness and maintaining its one-sided error.

Description of A'

The new non-adaptive algorithm works as follows. On input x, choose a random $y \in Y$. Query x on all indices A would have queried y, and output YES if the partial view of x is consistent with some $y' \in Y$. We give a formal description of the new algorithm in Algorithm 2.

Analysis

The algorithm A' is non-adaptive, as its queries depend only on its internal randomness (the choice of Γ and y). It has the same query complexity as A, since it uses the same decision trees. The algorithm has one-sided error since if $x \in Y$, we can take $y' = x$ at the last step of the algorithm, and $x|_I = y'|_I$ for every I.

Algorithm 2 Non-adaptive local algorithm, A'.

Input: Oracle access to $x \in \{0,1\}^n$.
Output: YES or NO.

1. Choose $\Gamma \sim D$.
2. Choose $y \in Y$ uniformly at random.
3. Let $\ell = (I, \sigma) = \Gamma(y)$.
4. Query x on the indices I.
5. Output "YES" if $\exists y' \in Y$ such that $x|_I = y'|_I$, and "NO" otherwise.

We are left with proving that the transformation does not decrease the soundness error. Towards this end, we relate the acceptance probability of A' to the *average* acceptance probability of A.

▷ Claim 21. For every $x \in \{0,1\}^n$:

$$\Pr[A'(x) = \text{YES}] = \mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]]$$

where the probabilities are over the internal randomness of A and A'.

This claim shows that soundness is maintained. If $x \in N$, then since (Y, N) is additive, $x + y \in N$ for every $y \in Y$. Hence, by using the soundness of A for *every* $x + y$, we get that

$$\Pr[A'(x) = \text{YES}] = \mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]] < \epsilon_{\mathrm{N}}$$ ◀

Proof of Claim 21. We begin by calculating $\mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]]$. Identifying the output YES with 1, we can take expectation instead of probability. We denote by $\Gamma(Y)$ the set of leaves in Γ labeled YES[9] , and get:

$$\begin{aligned}\mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]] &= \mathop{\mathbb{E}}_{y \in Y}\mathop{\mathbb{E}}_{\Gamma \sim D}[A(x+y)] = \mathop{\mathbb{E}}_{\Gamma \sim D}\mathop{\mathbb{E}}_{y \in Y}[A(x+y)] \\ &= \mathop{\mathbb{E}}_{\Gamma \sim D}\left[\frac{1}{|Y|} \cdot \sum_{\ell \in \Gamma(Y)} |(x+Y) \cap \Gamma^{-1}(\ell)|\right] \qquad (2)\end{aligned}$$

where in the last equality, we take into account all $y \in Y$ by iterating over each leaf $\ell \in \Gamma(Y)$ (for other leaves the algorithm's output is 0) and counting the number of y values for which $\Gamma(x+y) = \ell$ (i.e., that lead the algorithm to output the label of ℓ).

On the other hand, for any $I \subseteq \{0,1\}^n$ and $x \in \{0,1\}^n$ define:

$$H_I(x) = \begin{cases} 1, & \text{if } \exists y \in Y \text{ such that } x|_I = y|_I \\ 0, & \text{otherwise.} \end{cases}$$

With this notation, the last step of A' can be described as "output $H_I(x)$".

[9] This definition is equivalent to $\Gamma(Y) = \{\Gamma(y) \mid y \in Y\}$ since, as discussed above, a leaf ℓ is labeled YES if and only if there exists $y \in Y$ such that $\Gamma(y) = \ell$.

We get:

$$\Pr[A'(x) = \text{YES}] = \mathbb{E}[A'(x)] = \mathop{\mathbb{E}}_{\Gamma\sim D} \mathop{\mathbb{E}}_{\substack{y\in Y\\(I,\sigma)\leftarrow\Gamma(y)}} [H_I(x)]$$

$$= \mathop{\mathbb{E}}_{\Gamma\sim D}\left[\frac{1}{|Y|}\cdot \sum_{\ell=(I,\sigma)\in\Gamma(Y)} |Y\cap\Gamma^{-1}(\ell)|\cdot H_I(x)\right] \tag{3}$$

Here it is sufficient to iterate over the leaves in $\Gamma(Y)$: the algorithm A never errs on YES instances, so if ℓ is labeled NO, there cannot be $y \in Y$ such that $\Gamma(y) = \ell$.

From equations 2 and 3 it is enough to show that for every $\ell = (I, \sigma) \in \Gamma(Y)$:

$$|(x + Y) \cap \Gamma^{-1}(\ell)| = |Y \cap \Gamma^{-1}(\ell)| \cdot H_I(x)$$

Since (I, σ) is labeled YES, and as discussed above, there exists $y' \in Y$ such that $y'|_I = \sigma$ and $Y \cap \Gamma^{-1}(\ell)$ is not empty.

Consider the case $H_I(x) = 0$. We claim $(x + Y) \cap \Gamma^{-1}(\ell)$ is empty and hence the equality holds. Assume towards contradiction that this set is not empty. Then there exists $(x+y) \in (x+Y)$ such that $(x+y)|_I = \sigma$. Hence $(x+y)|_I = y'|_I$ and $x|_I = (y'-y)|_I$, which implies $H_I(x) = 1$ (since $y' - y \in Y$).

Next, consider the case $H_I(x) = 1$. We argue that $(x + Y) \cap \Gamma^{-1}(\ell)$ is not empty. $H_I(x) = 1$ implies there exists $y \in Y$ such that $x|_I = y|_I$, and $(x - y)|_I = 0$. Now $(x-y+y')|_I = (x-y)|_I + y'|_I = \sigma$, and hence $x-y+y' \in (x+Y)\cap\Gamma^{-1}(\ell)$ (as $-y+y' \in Y$). Since $(x + Y) \cap \Gamma^{-1}(\ell)$ is not empty, from Lemma 17 we get that $|(x + Y) \cap \Gamma^{-1}(\ell)| = |U|$. The set $Y \cap \Gamma^{-1}(\ell)$ is also not empty, and again from Lemma 17 $|Y \cap \Gamma^{-1}(\ell)| = |U|$. We conclude that $|(x + Y) \cap \Gamma^{-1}(\ell)| = |Y \cap \Gamma^{-1}(\ell)|$. ◁

▶ Remark 22. As in the previous reduction, the adaptive to non-adaptive reduction might be expansive in the terms of time complexity. Similar to before, sampling a YES instance uniformly at random might appear computationally intensive, although it can be implemented in polynomial time without much difficulty.

Unfortunately, determining whether there exists a YES instance consistent with a partial view of the input (step 5 in Algorithm 2) is computationally hard. We currently do not see a feasible way to perform this task efficiently, for reasons similar to those outlined in Remark 19. We leave addressing the efficiency of this reduction as an open problem.

References

1 Omar Alrabiah, Venkatesan Guruswami, Pravesh Kothari, and Peter Manohar. A near-cubic lower bound for 3-query locally decodable codes from semirandom CSP refutation. *Electron. Colloquium Comput. Complex.*, TR22-101, 2022.

2 Vahid R. Asadi and Igor Shinkar. Relaxed locally correctable codes with improved parameters. In *ICALP*, volume 198 of *LIPIcs*, pages 18:1–18:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 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.

4 Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3cnf properties are hard to test. In *STOC*, pages 345–354. ACM, 2003.

5 Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

6 Alessandro Chiesa, Tom Gur, and Igor Shinkar. Relaxed locally correctable codes with nearly-linear block length and constant query complexity. In *SODA*, pages 1395–1411. SIAM, 2020.

7 Gil Cohen and Tal Yankovitz. Relaxed locally decodable and correctable codes: Beyond tensoring. *Electron. Colloquium Comput. Complex.*, TR22-045, 2022. arXiv:TR22-045.

8 Gil Cohen and Tal Yankovitz. Asymptotically-good RLCCs with $(\log(n))^{2+o(1)}$ queries. *Electron. Colloquium Comput. Complex.*, TR23-110, 2023.

9 Marcel de Sena Dall'Agnol, Tom Gur, and Oded Lachish. A structural theorem for local algorithms with applications to coding, testing, and privacy. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1651–1665. SIAM, 2021. doi:10.1137/1.9781611976465.100.

10 Oded Goldreich. On the lower bound on the length of relaxed locally decodable codes. *Electron. Colloquium Comput. Complex.*, TR23-064, 2023.

11 Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In *CCC*, volume 33 of *LIPIcs*, pages 1–41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

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

13 Tom Gur and Oded Lachish. On the power of relaxed local decoding algorithms. In *SODA*, pages 1377–1394. SIAM, 2020.

14 Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Electron. Colloquium Comput. Complex.*, TR17-143, 2017.

15 Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Theory Comput.*, 16:1–68, 2020. doi:10.4086/toc.2020.v016a018.

16 Richard Wesley Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

17 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86. ACM, 2000. doi:10.1145/335305.335315.

18 Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *STOC*, pages 106–115. ACM, 2003.

19 Swastik Kopparty and Shubhangi Saraf. Local testing and decoding of high-rate error-correcting codes. *Electron. Colloquium Comput. Complex.*, TR17-126, 2017.

20 Vinayak M. Kumar and Geoffrey Mon. Relaxed local correctability from local testing. *CoRR*, abs/2306.17035, 2023.

21 Richard J. Lipton. Efficient checking of computations. In *STACS*, volume 415 of *Lecture Notes in Computer Science*, pages 207–215. Springer, 1990.

22 Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length. *IACR Cryptol. ePrint Arch.*, page 1062, 2019.

23 C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949. doi:10.1109/JRPROC.1949.232969.

24 David P. Woodruff. New lower bounds for general locally decodable codes. *Electron. Colloquium Comput. Complex.*, TR07-006, 2007.

25 David P. Woodruff. A quadratic lower bound for three-query linear locally decodable codes over any field. *J. Comput. Sci. Technol.*, 27(4):678–686, 2012.

26 Sergey Yekhanin. Locally decodable codes. *Found. Trends Theor. Comput. Sci.*, 6(3):139–255, 2012.