Electronic Colloquium on Computational Complexity, Report No. 132 (2025)

Time and Space Efficient Deterministic List Decoding

Joshua Cook* Dana Moshkovitz†

September 5, 2025

Abstract

Error correcting codes encode messages by codewords in such a way that even if some of the codeword is corrupted, the message can be decoded. Typical decoding algorithms for error correcting codes either use linear space or quadratic time. A natural question is whether codes can be decoded in near-linear time and sub-linear space simultaneously. A recent result by Cook and Moshkovitz gave efficient decoders that can uniquely decode Reed-Muller and other codes from a constant fraction (less than half) of corruption.

In this work, we address the problem of *list decoding* in near-linear time and sub-linear space. In the list decoding setting, most of the codeword is corrupted, and one wants to output a short list of potential messages that contains the true message. For any constants $\gamma, \tau > 0$, we give decoders for Reed-Muller codes that can decode from $1 - \gamma$ fraction of corruptions in time $n^{1+\tau}$ and space n^{τ}.

Our decoders work by extending the iterative correction technique of Cook and Moshkovitz. However, that technique, which gradually decreases the number of corruptions in the message, was tailored to the unique decoding setting. We first identify an intermediate problem, *codewords list recovery*, for which we can make iterative correction work. We then show how to reduce general list decoding to the codewords list recovery problem in efficient time and space. The reduction relies on local correction and testing. In the codewords list recovery problem, the input consists of n unordered lists of symbols from L codewords, where a small fraction of the lists is corrupted. The goal is to find the n unordered lists of symbols that are exactly the symbols from the L codewords.

In addition, we prove that any linear code with time-space efficient encoding or decoding must be local, in the sense that the codewords satisfy a local linear constraint. This rules out codes like Reed-Solomon from having time-space efficient encoding or decoding.

*jac22855@utexas.edu. Department of Computer Science, UT Austin. This material is based upon work supported by the National Science Foundation under grant number 2200956.

†danama@cs.utexas.edu. Department of Computer Science, UT Austin. This material is based upon work supported by the National Science Foundation under grant numbers 2200956 and 2312573.

ISSN 1433-8092

Contents

1 Introduction

1.1 Time-Space Efficient Codes

Error correcting codes encode information in a robust way, so that even if part of an encoding gets corrupted, it is still possible to *decode* the information. Codes are of great importance in communication and storage and have been studied for decades [Ham50; Ree00]. In theoretical computer science, codes have numerous applications, e.g., for pseudorandomness [KU06; STV01; Uma03; Gol10], interactive protocols [Lun+90; GKR15; RRR16], probabilistically checkable proofs [Bab+91; Fei+91; AS98; Aro+98] and cryptography [Tre04; McE78; Cou01; OS09; Sha79].

Definition 1.1 (Error correcting code)**.** *An error correcting code is a function $C : \Sigma^k \to \Sigma^n$ where different messages $x \neq x' \in \Sigma^k$ are mapped to codewords $C(x), C(x')$ that differ much: $C(x)_i \neq C(x')_i$ for at least d of $i \in [n]$. The* relative distance *of the code is d/n. The* rate *of the code is k/n. The alphabet of the code is Σ. An asymptotically good code is an infinite family $\{C_k\}_{k\geq 1}$ where $C_k : \Sigma^k \to \Sigma^n$ has constant rate, relative distance, and alphabet for all k.*

The number of indices $i \in [n]$ where two strings $w, w' \in \Sigma^n$ differ, i.e., $w_i \neq w'_i$, is called the *Hamming distance* between the strings. The number of indices i where $w_i = w'_i$ is the *agreement* of the strings. The process of computing $C(x)$ for a message x is called *encoding*. The process of computing x from a word w that agrees with a codeword $C(x)$ on most indices, is called *decoding*. Note that only words w that agree with a codeword $C(x)$ on all but less than $d/2$ indices can be uniquely decoded. An important generalization of decoding is *list decoding* [Eli57; Woz58]. Here, the decoder outputs all the codewords that have sufficient agreement with w. List decoding with short lists is often possible even for corrupted codewords with close to d errors (!) Such words may only agree with a codeword on a small fraction of the positions $i \in [n]$. List decoding turns out to be important for communication, and is crucial for many applications in theoretical computer science, such as pseudorandom generators [STV01; SU05], low error PCPs [MR08; DH09; IKW09; MZ24] and optimal inapproximability [Hås01].

Definition 1.2 (List decoding)**.** *Let $C : \Sigma^k \to \Sigma^n$ be an error correcting code. A γ-list decoding algorithm gets as input $w \in \Sigma^n$ and outputs all the codewords $C(x)$ with γn agreement withe w, i.e., $C(x)_i = w_i$ for at least γn indices $i \in [n]$. We call $n(1-\gamma)$ the list decoding radius and $1-\gamma$ the relative list decoding radius.*

Algorithmic coding theory aims to implement error correcting codes efficiently. Much work focuses on the time complexity of encoding and decoding, resulting in explicit asymptotically good codes that can be encoded and decoded in deterministic linear time [Spi95; SS96]. List decoding too can be achieved in linear time [GI03]. However, these algorithms have a linear space complexity. Other work focused on space complexity. All asymptotically good linear codes can be encoded in logarithmic space and quadratic time. Some codes (e.g., [Spi95]) were shown to also have decoding in logarithmic space, however the algorithms use large polynomial time. For instance, Spielman codes [Spi95] have uniform circuits with linear size and $d = O(\log(n))$ depth circuits that decode them. These low depth circuits have a straightforward algorithm evaluating them in $O(d) = O(\log(n))$ space (where $d > 2\log(n)$), but it runs in time $\Omega(2^d) = \Omega(n^2)$.

The current work focuses on deterministic algorithms that run *simultaneously* in almost linear time and sub-linear space. The question of whether there are explicit asymptotically good codes with time-space efficient encoding or decoding was raised in [BMS09] and [Gro06], respectively. Much of the initial work in the area focused on negative results. Bazzi, Mahdian and Spielman [BMS09] considered Turbo-like codes that could potentially have linear time and logarithmic space encoding, but showed that they were not asymptotically good. Bazzi and Mitter [BM05] proved that codes that can be encoded in *linear* time and sub-linear space cannot be asymptotically good. Gronemeier [Gro06] showed that deterministic *non-adaptive* decoders in almost linear time must run in near-linear space. In sharp contrast, recent work gave positive results. The paper [CM24] constructs explicit asymptotically good codes with deterministic almost-linear time and sub-linear space encoders [CM24]. It evades the impossibility result by considering almost linear time instead of linear time. The paper [CM25] constructs explicit asymptotically good codes with deterministic almost-linear time and sub-linear space decoders. It evades the impossibility result by using an adaptive decoder.

A downside of the paper [CM25] is that it only gives unique decoding algorithms. The goal of the current paper is to design time-space efficient deterministic list decoding algorithms. Our main theorem is as follows:

Theorem 1.3 (Good Codes With Time-Space Efficient Uniform List Decoding (Informal; See Section 7))**.** *For any constants $\tau > 0$ and $\gamma > 0$, there exists an explicit asymptotically good code with a deterministic γ-list decoding algorithm that runs in time $n^{1+\tau}$ and space n^{τ} on input of size n.*

The code for which we can give time-space efficient list decoding is the Reed-Muller code, as well as small alphabet versions of it obtained by concatenation with constant alphabet codes. The Reed-Muller code is natural and widely used. Its codewords are the truth tables of low degree multivariate polynomials over a finite field. The code is known for its strong local testing and correction properties, and those properties are crucial for our result.

We remark that if one only wants a *non-uniform* list decoder for codes that are locally testable and correctable (with weaker parameters than Reed-Muller) one can employ a technique from the paper [GM16] (See Section A in the appendix).

1.2 Overview of Our List Decoding Algorithm

Our main technical contribution is generalizing the unique decoding algorithm from the paper [CM25] to the list decoding setting. The method of [CM25] iteratively corrects the received word until it becomes a codeword. This approach makes sense in the unique decoding setting, but does not seem appropriate for the list decoding setting, where the received word might have small agreement with many different codewords.

The key idea of our solution is as follows. First, we identify a simpler problem that we call *codewords list recovery*, for which we are able to generalize the approach of [CM25]. Then, we show how to reduce the list decoding problem to the codewords list recovery problem in efficient time and space.

The general structure of our list decoding algorithm is described in Figure 1.

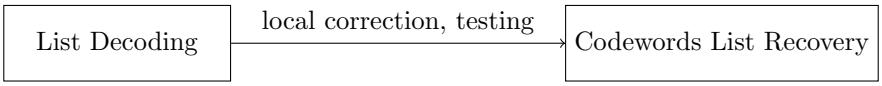

Figure 1: Codewords list recovery is solved in efficient time-space via an iterative correction algorithm (technique from [CM25]; suitably generalized). List decoding is solved via a time-space efficient reduction to codewords list recovery. Both the algorithm and the reduction use local correction. The reduction uses local testing too.

Codewords list recovery is a special case of the *list recovery* problem. In list recovery the input consists of short lists of possible symbols for each position in the codeword, and the goal is to find all the codewords that are consistent with most of the lists. In codewords list recovery, the lists come from ℓ codewords. Most of the lists are *exactly* consistent with those ℓ codewords, and the goal is to correct those lists:

Definition 1.4 (Codewords List Recovery)**.** *Let $\mathcal{C} : \Sigma^k \to \Sigma^n$ be a code. Let $Q^1, \ldots, Q^\ell \in \mathcal{C}(\Sigma^k)$ be codewords. For $i = 1, \ldots, n$ let $Q_i = \{Q_i^1, \ldots, Q_i^\ell\}$ be the set of codeword symbols in position i. The input is a sequence of lists $W_1, \ldots, W_n \subseteq \Sigma$, where for all indices except at most δn indices $i \in [n]$ we have $W_i = Q_i$. The goal is to find $Q^1, \ldots, Q^\ell$.*

Our algorithm for codewords list recovery requires a generalization of the algorithm in [CM25] to the case where there are l symbols rather than one per index. Meanwhile, it is the time-space efficient reduction from list decoding to codewords list recovery that does much of the heavy lifting for the list decoding algorithm. The reduction uses local correction to compose the lists and local testing to ensure that most lists match *exactly* with global codewords.

1.3 Time-Space Efficiency and Locality

Our list decoding algorithm relies on local correction and testing. The paper [CM25] on unique decoding relies on local correction. The codewords in the paper [CM24] or in Turbo-like codes [BMS09] have symbols

that depend on a small number of other codeword symbols and input symbols. A natural question is whether the locality of all those codes is an artifact of the ideas that were used so far in this area, or whether locality is somehow inherent to time-space efficient error correcting codes.

Perhaps surprisingly, we show that locality is inherent, in the following sense. We consider linear codes, i.e., linear transformations $\mathcal{C} : \Sigma^k \to \Sigma^n$. Moreover, we assume that $\mathcal{C}$ is *systematic*, i.e., $\mathcal{C}$ maps $x \in \Sigma^k$ to a codeword $C(x) = (x, y) \in \Sigma^n$ that starts with the input x. (Note that there is no loss of generality in considering systematic codes, and this assumption will allow us to handle encoding and decoding at the same time.) For linear codes, $\mathcal{C}(\Sigma^k)$ is a linear subspace, and the orthogonal subspace $\mathcal{C}(\Sigma^k)^\perp$ consists of the linear constraints that all the codewords satisfy. The *weight* of a constraint in $\mathcal{C}(\Sigma^k)^\perp$ is its Hamming distance from $\vec{0}$. The minimum weight over all non-zero elements in $\mathcal{C}(\Sigma^k)^\perp$ is a measure of the locality of $\mathcal{C}$. All locally testable and locally correctable codes have low weight constraints, however this is a necessary but not a sufficient condition for local testability and local correction.

We show that linear, systematic, codes $\mathcal{C}$ with time-space efficient encoding or decoding must have a low weight linear constraint:

Theorem 1.5 (Linear Codes Without Local Constraints Can Not Be Encoded Or Decoded Efficiently)**.** *Let $C : \Sigma^k \to \Sigma^n$ be a linear systematic code. Let t be the minimum weight of a non-zero element in $C(\Sigma^k)^\perp$. Then, any encoder for C running in time T and space S must have $ST = \Omega((n-k)(t-S))$. Further, any corrector for C that can correct d errors running in time T and space S must have $ST = \Omega(d(t-S))$.*

This theorem implies that codes with time-space efficient encoding or decoding must be local in the sense of having a low weight constraint. Locally correctable codes like those considered in this paper or in [CM25] indeed have low weight constraints. The codes considered in [CM24; BMS09] are not locally correctable but they have low weight constraints too.

The theorem rules out many codes from having time-space efficient encoding or decoding. For instance, the Reed-Solomon code does not have low weight constraints, and therefore cannot have time-space efficient encoding or decoding:

Corollary 1.6 (Reed-Solomon Codes Are Not Efficiently Encodable Or Decodable)**.** *Let $\mathbb{F}$ be a field and $\mathsf{deg} < |\mathbb{F}|$ be an integer. Then any encoder for the Reed-Solomon code, where the codewords are the tables of univariate polynomials of degree at most deg over $\mathbb{F}$, requires time T and space S such that $ST = \Omega((|\mathbb{F}| - \mathsf{deg} - 1)(\mathsf{deg} + 1 - S))$. Similarly, any decoder correcting d errors for the Reed-Solomon code requires time T and space S such that $ST = \Omega(d(\mathsf{deg} + 1 - S))$.*

2 List Decoding Main Ideas

In this section, we explain the main ideas that go into our list decoding algorithm. We discuss the connection between local decoding and *randomized* time-space efficient decoding. We explain the iterative correction technique of [CM25]. We recall the details of local correction for Reed-Muller codes and the main results about its local testing. Finally, we sketch our new algorithms: the algorithm for codewords list recovery and the reduction from list decoding to codewords list recovery. Our goal in this section is to review the main ideas so the reader can quickly grasp the information. Full details about the algorithms and their analysis appear in the body of the paper, and the reader is advised to consult them while reading this review.

2.1 Time-Space Efficient Unique Decoding

Observe that *randomized* time-space efficient unique decoding follows from *locally decodable codes* with a sub-linear number of queries [KT00; Yek12]. In locally decodable codes it is possible to get "random access" to the message given a corrupted codeword. That is, to decode a single symbol in the message x_i, it suffices to make a small number of queries to the corrupted codeword, chosen in a randomized fashion.

Definition 2.1 (Locally decodable code)**.** *A code $C : \Sigma^k \to \Sigma^n$ is locally decodable with q queries, ε decoding error, and δ relative decoding radius if it has a randomized decoder D that does the following. D is given oracle access to a word $w \in \Sigma^n$ and an index in the message $i \in [k]$, makes q queries to w, and outputs $b \in \Sigma$. If w is close to a codeword, that is, $w_j = C(x)_j$ for at least $1 - \delta$ fraction of $j \in [n]$, then for all $i \in [k]$ the decoder decodes the i'th symbol $b = x_i$ with probability at least $1 - \varepsilon$ over its randomness.*

By repeating the local decoder $\Theta(\log k)$ times, its error probability is sufficiently smaller than $1/k$, and running the decoder on every $i \in [k]$, one obtains a time-space efficient *randomized* decoder. Provided that the local decoder makes $q = n^{o(1)}$ queries and runs in time and space that are polynomial in q, the space complexity of the decoder we get is sub-linear and the time complexity is almost linear. Those ideas can also be extended to get a randomized time-space efficient list decoding algorithm from locally list decodable codes.

The challenge in decoding is to get time-space efficiency in a *deterministic* algorithm. A natural derandomization would take at least quadratic time: it would enumerate over all possible randomness strings of the local decoder (at least a linear number). For each randomness string, it would compute, in at least linear time, the corresponding codeword and check how close it is to the received word.

In contrast, we want a derandomization that does not increase the run-time substantially. Randomized local (i.e., sub-linear time) decoding provably cannot be derandomized without increasing the run-time significantly (to linear run-time), since any deterministic decoder must read the entire received word. However, the paper [CM25] succeeds in derandomizing (global) decoding without significantly increasing the runtime. Unfortunately, the derandomization requires the unique decoding setup.

The paper [CM25] assumes that the code is *locally correctable*. Local correction is similar to local decoding, except that the corrector receives $i \in [n]$ and wishes to find $C(x)_i$.

Definition 2.2 (Locally correctable code)**.** *A code $C : \Sigma^k \to \Sigma^n$ is locally correctable with q queries, ε decoding error, and δ relative decoding radius if it has a randomized decoder D that does the following. D is given oracle access to a word $w \in \Sigma^n$ and an index in the message $i \in [n]$, makes q queries to w, and outputs $b \in \Sigma$. If w is close to a codeword, that is, $w_j = C(x)_j$ for at least $1 - \delta$ fraction of $j \in [n]$, then for all $i \in [n]$ the decoder decodes the i'th symbol $b = C(x)_i$ with probability at least $1 - \varepsilon$ over its randomness.*

The approach of [CM25] is to iteratively decrease the number of corruptions in the received word by local correction. The insight is to decrease the number of corruptions only by a factor $n^{o(1)}$ in each iteration, and show that only $n^{o(1)}$ randomness strings suffice for such a small improvement. Since one can test each randomness string in linear time (comparing the corrected word with the received word), one gets time $n^{1+o(1)}$ overall.

2.2 Algorithm For Codewords List Recovery

The idea of iterative correction worked in the unique decoding setting, where there is a single codeword we wish to correct, and there is a natural progress measure, namely the distance to that codeword. In this paper, we generalize this approach to the case of the codewords list recovery problem (See Definition 1.4).

The input to the codewords list recovery problem consists of n sets of up to ℓ symbols each, denoted $W_1, \ldots, W_n \subseteq \Sigma$.

It is guaranteed that there are ℓ global codewords $Q^1, \ldots, Q^\ell$ such that all sets W_i, except perhaps δn, are consistent with the global codewords, i.e., $W_i = \{Q_i^1, \ldots, Q_i^\ell\}$. We refer to the $i \in [n]$ where there is no consistency as a *corruption*. The goal of the algorithm is to find the ℓ global codewords $Q^1, \ldots, Q^\ell$.

Note that for $\ell = 1$ this problem is precisely the unique decoding problem. We generalize the unique decoding algorithm of [CM25] from $\ell = 1$ to general ℓ. The algorithm is described in detail in Section 5.

The algorithm gradually decreases the number δn of corrupted lists from a small constant δ until it reaches $\delta < 1/n$ (i.e., no corruptions). When all lists are consistent with the ℓ codewords, there is a time and space efficient algorithm for identifying the codewords that give those lists (see Lemma 5.9). As in the unique decoding case [CM25], the algorithm decreases the number of corrupted lists δn using local correction, except that now local correction manipulates sets of l symbols instead of individual symbols. We will elaborate on local correction of the Reed-Muller code, and how to manipulate ℓ symbols, in the next subsection. Overall, we get an almost-linear time and sub-linear space algorithm for the codewords list recovery problem of Reed-Muller codes.

2.3 Local Correction for Reed-Muller Codes

In this subsection we discuss local correction in the list decoding regime, which we refer to as *local list correction*. Recall that we discussed local correction in the unique decoding regime in Section 2.1. Local list

correction is crucial both for the algorithm for codewords list recovery, and for the reduction of list decoding to codewords list recovery.

There are several ways to define local correction in the list decoding regime, so first we discuss the definition we use and contrast it with another standard definition. The corrector is given oracle access to a highly corrupted codeword. Let l denote the number of codewords in the γ-list decoding of the received word. On input an index $i \in [n]$ the corrector wishes to find the i'th symbol of each of the l codewords in the list decoding using a small number of queries to the word. One standard definition of local list correctors involves outputting l different circuits, each consistent with a single codeword. On input $i \in [n]$ the j'th circuit outputs the i'th symbol of the j'th codeword. We use a weaker definition, where the corrector only needs to output at most l symbols for the i'th index, containing the symbols from the l codewords, but without associating each with its codeword. The definition we use for local correction is as follows.

Definition 2.3 (Locally list correctable code (weak definition))**.** *A code $C : \Sigma^k \to \Sigma^n$ is locally list correctable with q queries, ε correction error, γ agreement, and l list size, if it has a randomized decoder D as follows. D is given access to a word $w \in \Sigma^n$ and an index in the codeword $i \in [n]$, makes q queries to w, and outputs $L_i \subseteq \Sigma$, $|L_i| \leq l$, such that if w is close to a codeword, that is, $w_j = C(x)_j$ for at least γ fraction of $j \in [n]$, then for every $i \in [n]$ the decoder's list L_i contains $C(x)_i$ with probability at least $1 - \varepsilon$ over its randomness.*

We focus on the Reed-Muller code, which is perhaps the most natural locally correctable code (See also the survey by Yekhanin [Yek12] on local decoding). First, let us formally define the Reed-Muller code: For natural numbers $\mathsf{dim}, \mathsf{deg} \geq 1$ and a finite field $\mathbb{F}$ the degree deg Reed-Muller code over $\mathbb{F}^{\mathsf{dim}}$ is the set of dim-variate, total degree deg polynomials. The codeword length is $n = |\mathbb{F}^{\mathsf{dim}}|$, the alphabet is $\mathbb{F}$, and we identify the indices $i \in [n]$ with points $x \in \mathbb{F}^{\mathsf{dim}}$. The codeword consists of the evaluations of the polynomial on all points in $\mathbb{F}^{\mathsf{dim}}$. That is, for a polynomial $p : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$, the corresponding codeword is $(p(x))_{x \in \mathbb{F}^{\mathsf{dim}}}$.

The Reed-Muller code has a simple local list correction [AS97; STV01], which we describe next. Given a received word $w : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$ and a point to correct $x \in \mathbb{F}^{\mathsf{dim}}$, local correction is as follows:

1. Pick a line through x in $\mathbb{F}^{\mathsf{dim}}$ uniformly at random.
2. Perform Reed-Solomon list decoding [Sud97; GS99; FB98] of the restriction of w to the line to find degree-deg polynomials with at least $\approx \gamma$ agreement.
3. Output the evaluations of all the polynomials from step 2 on x.

The corrector only queries $|\mathbb{F}|$ points out of $n = |\mathbb{F}^{\mathsf{dim}}|$, and is therefore local. Given access to w which is a codeword Q, the corrector always outputs $Q(x)$. The idea is that if w agrees with a polynomial Q on a large fraction of the points in $\mathbb{F}^{\mathsf{dim}}$, then a uniform line is expected to contain a similar fraction of points they agree on. Therefore, $Q(x)$ would probably be outputted. On the other hand, if there are few $x \in \mathbb{F}^{\mathsf{dim}}$ such that $w(x) \notin \{Q^1(x), \ldots, Q^l(x)\}$ where $Q^1, \ldots, Q^l$ is the list decoding of w, a random line will rarely contain enough of those to output a symbol inconsistent with $Q^1, \ldots, Q^l$.

The property of lines that enables local correction is *sampling*:

Definition 2.4 (Sampling)**.** *A family of samples $\{S\}$ where each sample S is a size-s subset of $\mathbb{F}^{\mathsf{dim}}$, is a sampler with accuracy error ϵ and strong confidence error δ if for any $A \subseteq \mathbb{F}^{\mathsf{dim}}$, $\mu = |A|/|\mathbb{F}^{\mathsf{dim}}|$, for at least $1 - \delta\mu$ fraction of the samples S we have*

$$\left| \frac{|A \cap S|}{|S|} - \frac{|A|}{|\mathbb{F}^{\mathsf{dim}}|} \right| \leq \varepsilon.$$

In the context of local correction, the set A corresponds to corrupted symbols in w, or to points where w agrees with a codeword Q. Note the dependence of the error probability $\delta\mu$ on μ (the relative size of A). Hence, when μ corresponds to the fraction of corruptions in w, the sampling error is appropriately smaller.

Like [CM25], we will need the following variations on local correction, used here in the list decoding regime instead of the unique decoding regime:

Error reduction: For our iterative correction technique, it is imperative to decrease the error probability of the corrector below what a random line can obtain. Specifically, we wish to decrease the error by a factor larger than the number of queries. To do so, the corrector picks several lines through x instead of just one, and only picks values for x that repeat for the majority of lines. The paper [CM25] constructs an appropriate sampler: With probability $1 - \mu|\mathbb{F}|^{-\omega(1)}$ the majority of lines sample A within accuracy error $\epsilon = |\mathbb{F}|^{-\Omega(1)}$. See Section 5.2 for formal results.

Average-case local correction: We consider local correction that works for *most* points $x \in \mathbb{F}^{\mathsf{dim}}$, and not necessarily for *all* points. This will allow us to decrease the amount of randomness that the corrector uses from n per point to $n^{o(1)}$ per point. This decrease in randomness is crucial in order to obtain almost linear time and sub-linear space.

In the algorithm for codewords list recovery we will use a further variation on local correction, namely handling inputs that consist of ℓ symbols instead of a single symbol per position:

List recovery input: The corrector can be made to work for list recovery problems rather than list correction problems. Here the input is a *set* of ℓ values in $\mathbb{F}$ per $x \in \mathbb{F}^{\mathsf{dim}}$ as opposed to a single value in $\mathbb{F}$. In other words, the received word w can be thought of as a *multi-string*, rather than a string. The only change to the local corrector is performing Reed-Solomon list recovery instead of list decoding [KV03; Ale05; Gur07].

2.4 Local Testing

In the next subsection we describe the reduction of the list decoding problem to the codewords list recovery problem in efficient time-space. This reduction completes our list decoding algorithm. For the reduction, we crucially rely on incorporating local testing in our local correction. The use of a local testing theorem is new to the list decoding regime and did not appear in the time-space efficient unique decoding algorithm [CM25].

In this sub-section we discuss the definition of local testing we use and contrast it with other standard definitions. In local testing, there is oracle access to a word in Σ^n, which may or may not agree much with a codeword. The tester makes a small number of queries to the word and either accepts or rejects. It is supposed to accept codewords. Since the tester makes a small number of queries, it cannot distinguish words that agree with a codeword on almost all the indices from codewords. Hence, the tester is only expected to reject words that are far from the code.

The definition we use asks that the acceptance probability of the tester directly corresponds to the agreement of the word with a codeword, up to a small *additive* term:

Definition 2.5 (Locally Testable Codes (additive approximation))**.** *We say that a code $C : \Sigma^k \to \Sigma^n$ is a locally testable code (LTC) with q queries and additive approximation α if there is a tester that makes q queries to a received word $w \in \Sigma^n$ and either accepts or rejects such that:*

- *If $w = C(x)$, then the tester always accepts.*
- *If the tester accepts with probability β, then there is a codeword $C(x)$ that agrees with w on at least $\beta - \alpha$ and at most $\beta + \alpha$ fraction of positions.*

The additive approximation definition is much stronger than other definitions of local testing that are sometimes considered. Oftentimes, certain distance from the code implies a certain rejection probability, but farther distance from the code does not necessitate higher rejection probability. A different standard definition (Definition 4.22) requires that the probability that the tester rejects provides a multiplicative approximation (with a factor strictly larger than 1) of the distance to a codeword. This definition allows the distinguishing of codewords from words that have noticeable distance from the code, but does not provide useful information if w is very far from the code. The small additive approximation in Definition 2.5 means that one can detect codewords even if they agree with the received word on a very small fraction of positions.

A slight modification of the local corrector of the Reed-Muller code described in the previous section makes the corrector into a local tester with a small additive approximation $|\mathbb{F}|^{-\Omega(1)}$ [MR06]. As of the time

of writing this paper, we do not know of other codes with a randomness-efficient local corrector and tester with a small additive approximation.

In the subsection that follows we discuss the time-space efficient reduction from list decoding to codewords list recovery, and give more details about the reason for using local testing.

2.5 Reduction From List Decoding To Codewords List Recovery

In this subsection we describe the time-space efficient reduction from list decoding to codewords list recovery. This reduction is one of the key contributions of the current work. In the reduction we use the local list correction of subsection 2.3, combined with local testing as, described in subsection 2.4.

In the list decoding problem, we are given oracle access to a word $w : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$ and we wish to find all the dim-variate polynomials $Q^1, \ldots, Q^l$ of degree at most deg that agree with w on at least γ fraction of the points in $\mathbb{F}^{\mathsf{dim}}$. We wish to construct a codewords list recovery instance that corresponds to the list decoding of w.

We could invoke the local list corrector on each possible randomness to obtain a list recovery instance. The main difficulty is that this list recovery instance is not necessarily a *codewords* list recovery instance with a small number of corruptions. There are two ways the sets we generate could go wrong: One is sets missing symbols from certain nearby codewords. We call these *in-codeword errors*. Because of sampling (discussed in Subsection 2.3), in-codeword errors are rare. The other is sets that contain symbols that do not belong to any nearby codeword. We call these *out-of-codeword errors*. If out-of-codeword errors are rare as well, then we are done. However, it is possible that many indices $x \in \mathbb{F}^{\mathsf{dim}}$ have out-of-codeword errors, because many of the $n^{o(1)}$ lines through x are consistent with local polynomials that do not correspond to any global polynomials.

To make sure that out-of-codeword errors are rare, we adapt our local corrector to incorporate local testing (See Definition 2.5). Local testing implies that if there is significant local agreement for many random choices of the local correction, then the local agreement must correspond to agreement with a global polynomial Q^i. We use the local tester of [MR06] we mentioned before, and for that we change the local corrector to query $O(1)$-dimensional subspaces and not lines. Importantly, we only use subspaces for the reduction to list recovery. The rest of the correction uses more query efficient correctors.

3 Only Local Codes Are Time and Space Efficiently Encodable or Decodable

In this section we give the simple proof of Theorem 1.5 that shows that time-space efficient linear codes must have low weight constraints. The general result (see Theorem 3.6) is that if the uniform distribution over codewords is t-wise independent, then correcting d codeword symbol *erasures* requires an algorithm running in time T and space S such that $ST = \Omega((t-S)d)$. This implies the theorem since a linear code is t-wise independent if and only if it does not have a linear constraint involving t variables. General code correction implies erasure correction, so this also implies a lower bound for more general correction. Further, encoding a code is a special case of erasure correction, so this result is also a lower bound for encoding codes.

The basic proof strategy is to show that even conditioned on knowing S symbols of information about an input codeword, to output $S+1$ symbols that were erased, we need to read some block of $t-S$ symbols from the input word. Thus to write d symbols, one must use $(t-S)\frac{d}{S+1}$ time steps.

To show that even with S bits of information we need to read $t-S$ symbols of the codeword, we consider the state after reading the next $t-S-1$ symbols. If we only read $t-S-1$ symbols and then wrote $S+1$ symbols, then due to t-wise independence, there are t symbols of information in the symbols read plus the symbols written. Yet we only know the S symbols we began with plus the $t-S-1$ symbols we read, which is only $t-1$ symbols of information. The symbol of information missing must be in the $S+1$ symbols we wrote since we read the remaining $t-S-1$ symbols as plain text. Thus the corrector can not write $S+1$ symbols before it reads all $t-S$ symbols. Since the algorithm has bounded space, at any step it only has roughly S bits of information about the specific codeword.

In the rest of this section, we will formalize this argument.

3.1 Erasure Decoding

In this section, we will briefly define erasure decoding. An erasure of a word is just the same word with some of the symbols erased.

Definition 3.1 (Erasure)**.** *Take any integer n and alphabet Σ. Call some $\perp \notin \Sigma$ the empty symbol. We say that a function $E : \Sigma^n \to (\Sigma \cup \{\perp\})^n$ erases d symbols if for some set $H \subseteq [n]$ with $|H| = d$ we have that for all $x \in \Sigma^n$ that for all $i \in H$ that $E(x)_i = \perp$ and for all $j \notin H$ we have $E(x)_j = x_j$.*

Similarly, for any $x \in \Sigma^n$ and $x' \in (\Sigma \cup \{\perp\})^n$, we say that x' is x with d erasures, if for some E that erases d symbols we have that $E(x) = x'$.

The idea of erasure decoding is that instead of corruption being a symbol change, corruption is a symbol being replaced with a special empty symbol outside the original alphabet. In erasure decoding, the decoder knows what symbols are corrupted, but not what their original value was. In the standard definition of erasure decoding, one does not know what symbols will be erased before hand. Our results hold even if one does know which symbols will be erased. Finally, our results are stated most naturally in terms of correction, so we will define targeted erasure correction.

Definition 3.2 (Targeted Erasure Correction)**.** *Take any integer n and alphabet Σ. Let $E : \Sigma^n \to (\Sigma \cup \{\perp\})^n$ be a function that erases d symbols. Let $C : \Sigma^k \to \Sigma^n$ be a code. We say that a function $D : (\Sigma \cup \{\perp\})^n \to \Sigma^n$ is an erasure corrector for code C and error E if for all $x \in \Sigma^k$ we have that*

$$C(x) = D(E(C(x))).$$

Erasure decoding is easier than standard decoding because one can reduce erasure decoding to standard decoding by guessing the erased symbols.

Lemma 3.3 (Correctors Imply Targeted Erasure Correction)**.** *Take any integer n and alphabet Σ. Let $C : \Sigma^k \to \Sigma^n$ be a code. Suppose there is a time T space S, distance d corrector for C. Then for every E that erases d symbols, there is an erasure corrector for code C and error E running in time $O(T)$ and space $O(S)$.*

Proof. For any given input x' that is a d erasure of x, just replace every symbol missed from x' with a random symbol, call the result x^*. Now x^* is within d of x and we can use the decoder for C to recover x. □

An advantage to defining targeted erasure correction is that encoding is a special case of targeted erasure decoding (where none of the message bits are corrupted). Thus our decoding lower bounds apply to encoding as well.

Lemma 3.4 (Encoding is Targeted Erasure Correction)**.** *Take any integer n and alphabet Σ. Let $C : \Sigma^k \to \Sigma^n$ be a systematic code. Then the function C is an erasure corrector for code C and some function E that erases $n - k$ symbols.*

Proof. The function E just erases all but the k message symbols from the codeword. □

3.2 Proof of Theorem 1.5

Now we will give a slightly more formal overview of the proof. In our proof, it is useful to imagine that the corrector gets as input a uniformly random codeword, and is trying to read just enough of it to learn a few symbols. At any given point in time, the corrector has some state which is reached by some specific set of codewords. The first observations is that since there are few states, the state reached with high probability must be reached by many other codewords. In Lemma 3.5 we argue that as long as there are enough codewords that map to the current state, the corrector must read a lot of symbols before outputting the next $S + 1$ corrections. We then use Lemma 3.5 many times in Theorem 3.6 to give the final result.

Lemma 3.5 (Low Space t-Wise Independent Code Correctors Need Many Queries To Output A Few Symbols)**.** *Let $C : \Sigma^k \to \Sigma^n$ be a systematic code such that the uniform distribution over codewords in C is t-wise*

independent. Let E be a function which erases d symbols. Let $S < d$ be a constant and Y be a uniform distribution over at least $|\Sigma|^{k-S}$ codewords in C.

Take any deterministic algorithm D that takes as input words from $E(Y)$ and writes to $S+1$ distinct indexes erased by E such that for any $y \in Y$, any time D writes symbol i it writes y_i. In other words, D correctly corrects $S+1$ symbols erased by E from codewords in Y. Then with probability at most $\frac{1}{|\Sigma|}$ will corrector D read less than $t-S$ symbols.

Proof. First we will show that with probability at most $\frac{1}{|\Sigma|}$ will Y conditioned on the first $t-S-1$ symbols be a distribution $|\Sigma|^{k-t}$ or fewer codewords. Then we will argue that this implies that D fails on some input from Y if it reads less than $t-S$ symbols.

Conditioning Y on $t-S-1$ symbols partitions the at least $|\Sigma|^{k-S}$ codewords in Y is distributed over into $|\Sigma|^{t-S-1}$ groups. In the worst case, only $|\Sigma|^{k-S-1}$ elements could be in a partition with $|\Sigma|^{k-t}$ or fewer elements in it (since $|\Sigma|^{t-S-1}|\Sigma|^{k-t} = |\Sigma|^{k-S-1}$). Thus the probability that a random $y \in Y$ will be in such a partition is at most $\frac{|\Sigma|^{k-S-1}}{|\Sigma|^{k-S}} = \frac{1}{|\Sigma|}$. Thus with probability at most $\frac{1}{|\Sigma|}$ will Y conditioned on the first $t-S-1$ symbols read be a distribution over $|\Sigma|^{k-t}$ or fewer codewords.

Let Y' be Y conditioned on the first $t-S-1$ symbols read. Suppose that Y' is a distribution over more than $|\Sigma|^{k-t}$ codewords. We will now argue that if D writes $S+1$ symbols, no matter what $S+1$ symbols are written, there will be some input in the distribution of Y' such that the symbols written are incorrect. Consider the set of $S+1$ coordinates written and the $t-S-1$ read. If Y' restricted to these t coordinates were constant, then Y' must only be a distribution over at most $|\Sigma|^{k-t}$ codewords (the total number of codewords divided by the number of assignments to these t symbols) since the coordinates of C are t-wise independent. But Y' is a distribution over more than $|\Sigma|^{k-t}$ symbols, so Y' is not fixed on these t coordinates. However, it is fixed on the $t-S-1$ coordinates read, so it must not be fixed on the $S+1$ coordinates written. Therefore, whatever is written after reading these coordinates must be wrong on some codewords from Y'. □

Now applying this lemma several times gives us the main result of this appendix.

Theorem 3.6 (t-Wise Independent Code Correctors Require Large Time-Space)**.** *Let $C : \Sigma^k \to \Sigma^n$ be a systematic code such that the uniform distribution over codewords in C is t-wise independent. Let E be a function which erases d symbols. Let $S < d$ be a constant. Than any function $D : (\Sigma \cup \{\perp\})^n \to \Sigma^n$ running in time T and space $S\log(|\Sigma|)$ that is an erasure corrector for code C and error E must have $ST = \Omega((t-S)d)$.*

Proof. Let Y be the distribution over all codewords in C. Consider the expected runtime of D. At any given time step, the probability that Y conditioned on the state of the corrector is a distribution over $\frac{1}{|\Sigma|}\frac{|\Sigma|^k}{|\Sigma|^S} = |\Sigma|^{k-S-1}$ or fewer states is at most $\frac{1}{|\Sigma|}$, since $|\Sigma|^k$ is the number of codewords and $|\Sigma|^S$ is the number of states of the corrector. If Y conditioned on the current state is a distribution over more than $|\Sigma|^{k-S-1}$ states, by Lemma 3.5, the probability that the decoder outputs $S+2$ symbols before reading $t-S-1$ symbols is at most $\frac{1}{|\Sigma|}$. Thus, at any given time step, with probability at least $\left(1-\frac{1}{|\Sigma|}\right)^2 > \frac{1}{4}$ will the corrector read $t-S-1$ symbols before it outputs the next $S+2$ symbols. Thus in expectation, at any given step in the computation, writing the next $S+2$ symbols of output requires reading $\Omega(t-S)$ symbols of the input.

By splitting the d symbols to be corrected into $\Omega\left(\frac{d}{S+2}\right)$ chunks, we conclude that the expected final time of the decoder must be

$$\begin{aligned} T \geq& \Omega\left((t-S)\left(\frac{d}{S}\right)\right) \\ ST \geq& \Omega\left((t-S)d\right). \end{aligned}$$

□

Now we prove Theorem 1.5.

Proof. First we show that any t coordinates of a random linear codeword are independent if and only if there is no way to write a non-trivial linear constraint of the codeword using just those t coordinates. Since we could not find a reference for this folklore result, we give a brief proof here.

If there is a non-trivial linear constraint on t coordinates, then some choice of assignment to those coordinates is invalid. So a random codeword can not be independent because some assignments to those t happen with probability 0 instead of probability $\frac{1}{|\Sigma|^t}$.

For the other direction, first observe that evaluating a linear function on a uniformly random input gives a uniformly random distribution over it's image. If the image of the linear function on t coordinates is not uniform, then this means not all assignments to those t coordinates are possible. Thus the linear function that only outputs these t coordinates is not onto. Any linear subspace is defined by constraints, and the linear subspace that is the image on these t coordinates is not the whole space, so it has a linear constraint.

Now we can apply Lemma 3.5 to get that any specific, large erasure cannot be efficiently corrected. Now combining this with Lemma 3.4 and Lemma 3.3 we get our final result. □

Since self dual codes don't have small constraints, this recovers the result of [SV06] that self dual codes cannot be encoded time and space efficiently and generalizes it to decoding, and more general codes.

4 Preliminaries

4.1 Error Correcting Codes

Our results are about decoding error correcting codes. An error correcting code is a function that maps any two distinct messages to codewords that are far apart in Hamming distance. The Hamming distance of two strings is the number of indexes where they differ.

Definition 4.1 (Hamming Distance)**.** *For any alphabet Σ and integer n, we define the Hamming distance $\Delta : \Sigma^n \times \Sigma^n \to \mathbb{N}$ by*

$$\Delta(x, y) = |\{i \in [n] : x_i \neq y_i\}|.$$

Now we can define an error correcting code, often just called a code.

Definition 4.2 (Error Correcting Code)**.** *For alphabets Σ_1 and Σ_2, integers $k, n \in \mathbb{N}$ and distance $d \in \mathbb{N}$, a code is a function $C : \Sigma_1^k \to \Sigma_2^n$ such that for all $x, y \in \Sigma_1^k$ with $x \neq y$ we have that*

$$\Delta(C(x), C(y)) \geq d.$$

We call d the distance of C, Σ_2 the alphabet of the code, and Σ_1 the alphabet of the message. We call $\frac{k}{n}$ the rate of the code.

Any $x \in \Sigma_1^k$ we call a message and we call $C(x)$ its codeword. We define the codewords of C to be the set $\{C(x) : x \in \Sigma_1^k\}$.

We say the code is asymptotically good if $d = \Omega(n)$ and $n = O(k)$.

Often an error correcting code is defined in terms of its set of codewords and not its encoding function. In particular

Lemma 4.3 (Codes From Codewords)**.** *Suppose that there is a set $C' \subseteq \Sigma^n$ such that for all $u, v \in C$ with $u \neq v$ we have that $\Delta(u, v) \geq d$. Then there is a code $C : [|C'|] \to \Sigma^n$ with distance d such that the set of codewords for C is C'.*

We say that $y \in C$ if there exists some x such that $C(x) = y$.

This comes from ordering the elements of C' in an arbitrary order and letting $C(i)$ output the ith element.

The primary feature of codes we are interested in is decoding. First, we define unique decoding.

Definition 4.4 (Unique Decoding)**.** *Suppose we have a code $C : \Sigma_1^k \to \Sigma_2^n$. Then we say a function $D : \Sigma_2^n \to \Sigma_1^k$ is a decoder with decoding radius d if for all messages $x \in \Sigma_1^k$ and strings $y \in \Sigma_2^n$ with $\Delta(C(x), y) \leq d$ we have that $D(y) = x$. We call $\frac{d}{n}$ the relative decoding radius.*

If D can be computed in time T and space S, we say that C is decodable in time T and space S. We say C is time T and space S decodable if such a D exists.

In contrast to decoding, one can also ask for the codeword to be corrected. This is where our decoder, instead of outputting the message x, it instead outputs the codeword $C(x)$.

Definition 4.5 (Correcting)**.** *Suppose we have a code $C : \Sigma_1^k \to \Sigma_2^n$. Then we say a function $D : \Sigma_2^n \to \Sigma_2^n$ is a corrector with correcting radius d if for all messages $x \in \Sigma_1^k$ and strings $y \in \Sigma_2^n$ with $\Delta(C(x), y) \leq d$ we have that $D(y) = C(x)$. We call $\frac{d}{n}$ the relative correcting radius.*

If D can be computed in time T and space S, we say that C is correctable in time T and space S with a correcting radius of d.

A straight forward consequence of a code with minimum distance d is that any codeword can be corrected up to $d/2$ errors. This is because for a distance d code, any string can only have one codeword closer than $d/2$, otherwise there would be 2 codewords closer together than d. And this is tight: if there are two codewords at distance d, then the string halfway between them is distance $d/2$ from both. So one can only correct codewords uniquely up to half the distance of the codeword.

But if one allows us to output a few candidate codewords, then one can do better. In the last example, there are two codewords with distance $d/2$ from a point, but there are *only* two. So if we allow our corrector to output 2 strings, then one can do correction with distances greater than $d/2$. We call this list correction. A famous result in list correction is called the Johnson bound. It shows that as long as the distance we want to correct isn't too close to d, then there is a short list of codewords that are near any point.

4.2 Multi-strings

Another type of error correction is called list recovery. Sometimes during correction, one is not very confident about the exact symbols at each location in the string, but has a short list of candidate symbols at each location. This model allows the decoder to better quantify uncertainty as to what symbols are correct and can lead to better decoding results.

For our results, we frequently use string-like objects where each location is not a specific symbol, but a set of possible symbols. We call these objects multi-strings.

Definition 4.6 (Multi-string)**.** *. Let Σ be an alphabet. Denote $2^\Sigma = \{A \subseteq \Sigma\}$ to be the set of all subsets of Σ. For any natural number $\ell \leq |\Sigma|$ we denote $\binom{\Sigma}{\ell} = \{A \subseteq \Sigma : |A| \leq \ell\}$ to be the set of subsets of Σ of size at most ℓ.*

Let n be an integer. We call any $w \in (2^\Sigma)^n$ a multi-string over alphabet Σ of length n. Specifically, w is such that for each $i \in [n]$, $w_i \subseteq \Sigma$. We call w an ℓ-multi-string if for each $i \in [n]$ we have that $|w_i| \leq \ell$. Equivalently, w is an ℓ-multi-string if $w \in \binom{\Sigma}{\ell}^n$.

If w^1 or w^2 is a string in Σ^n, then we can also interpret it as the 1-multi-string such that for all $i \in [n]$ we have that $w_i = \{w_i\}$.

The use of $\binom{\Sigma}{\ell}^n$ to denote an ℓ-multi-string is common in the literature [HRZW17; Kop+23], but is often referred to as just a string. We prefer to emphasize that this is not a simple string and we will not manipulate them like simple strings, so we call them multi-strings to emphasize this. As an example, we will consider multi-strings that are taken as a union of codewords. This only makes sense if we view the coordinates of the multi-string as sets.

Definition 4.7 (Multi-string Union)**.** *For alphabet Σ and integer n, let $w^1, w^2 \in (2^\Sigma)^n$ be length n multi-strings. Then we define the union of w^1 and w^2 as the multi-string $w^1 \cup' w^2$ such that for all $i \in [n]$*

$$(w^1 \cup' w^2)_i = w_i^1 \cup w_i^2.$$

Similarly, for k multi-strings $w^1, \ldots, w^k$, we define $\bigcup'_{j\in[k]} w^j$ such that for all $i \in [n]$ we have $(\bigcup'_{j\in[k]} w^j)_i = \bigcup_{j\in[k]} w_i^j$.

For list recovery, we want to recover the set of codewords near a given multi-string. Further, as an intermediate step to outputting all codewords near a multi-string, we will first ask for the multi-string containing all the codewords near a multi-string. So for notation, we define the following.

Definition 4.8 (Near Codewords)**.** *For an alphabet Σ, integer n, code C whose codewords are in Σ^n, multi-string w, and agreement γ, we define the list of γ near codewords to w as the set of codewords that agree with w on γ fraction of indices.*

$$\mathsf{near}^C_\gamma(w) = \{y \in C : \Pr_{i\in[n]}[y_i \in w_i] \geq \gamma\}.$$

And we define the multi-string of γ near codewords to w as

$$\mathsf{near\text{-}ms}^C_\gamma(w) = \bigcup_{y\in \mathsf{near}^C_\gamma(w)}{}' y.$$

We say that ℓ-multi-string w is an ℓ-codeword multi-string if $\mathsf{near\text{-}ms}^C_1(w) = w$, or equivalently if for some set of codewords $Q_1, \ldots, Q_\ell$ we have $w = \bigcup'_{i\in[\ell]} Q_i$.

4.3 List Recovery

Now that we have introduced multi-strings, we can formally define list recovery.

Definition 4.9 (List Recovery)**.** *Suppose we have a code $C : \Sigma_1^k \to \Sigma_2^n$, and integers ℓ and L. Then we say a function $D : \binom{\Sigma_2}{\ell}^n \to (\Sigma_2^n)^L$ is a list recovery algorithm with list recovering radius d, input list length ℓ and output list length L if for all ℓ-multi-strings $w \in \binom{\Sigma_2}{\ell}^n$ and $y \in C$ such that $\Pr_i[y_i \notin w] \leq d$ we have that for some $j \in [L]$ that $D(w)_j = y$.*

If D can be computed in time T and space S, we say that C is list recoverable in time T and space S with a correcting radius of d.

Equivalently, we can rephrase list recovery to say that $\mathsf{near}^C_{1-d/n}(w) \subseteq D(w)$, abusing notation slightly if we interpret the output of D as a set instead of a list. Note that in the global correcting setting, we can check if one of the outputs of D is actually within distance d of w before outputting it, so we can actually assume that $\mathsf{near}^C_{1-d/n}(w) = D(w)$.

One may wonder whether most codes even have better list recovering radius than unique decoding radius. The famous Johnson bound shows that for large distance codes (distance close to 1), the answer is yes. We use a slight generalization of the Johnson bound [Joh62]. Similar extensions were given by Guruswami and Sudan [GS99]. For completeness, we include a proof.

Lemma 4.10 (Extension to The Johnson Bound)**.** *Suppose for alphabet Σ and integer n, there is a code C with distance d whose codewords are length n strings over Σ. Then for any constant $c > 1$ and integers a and ℓ such that $a > \sqrt{\ell n(n-d)c}$, and any $w \in \binom{\Sigma}{\ell}^n$ we have that for*

$$L = \ell \frac{n}{a} \frac{1}{1-1/c}$$

there are at most L strings in Σ^n that agree with w on more than a locations. That is, C is list recoverable with list recovery distance $n-a$, input list length ℓ, and output length L.

Proof. For multi-strings x and w, define agreement by $\mathsf{agr}(x, w) = \Pr_{i\in[n]}[x_i \subseteq w_i]$. For strings s^1 and s^2 this simplifies to $\mathsf{agr}(x, w) = \Pr_{i\in[n]}[s^1_i = s^2_i]$.

Suppose for some alphabet Σ, input list length ℓ, and string length n that we have $w \in \binom{\Sigma}{\ell}^n$. Suppose for some agreement a there are L strings $s^1, s^2, \ldots, s^L \in \Sigma^n$ such that for all $j \in [L]$ we have that the agreement between s^j and w is at least a: $\mathsf{agr}(s^j, w) \geq a$. Then the average over $i \in [n]$ and $\sigma \in w_i$ of the number of $j \in [L]$ such that $s^j_i = \sigma$ is at least $\frac{La}{\ell n}$. So on average, a random s^j agrees with at least $\frac{La}{\ell n} - 1$ other strings at any of the a indexes it agrees with w. Thus the average number of indexes s^j agrees with a random other s^k is at least this number times a divided by L. This is

$$\mathop{\mathbb{E}}_{j\neq i}[\mathsf{agr}(s^j, s^i)] \geq \left(\frac{La}{\ell n} - 1\right)\frac{a}{L} = \frac{a^2}{\ell n} - \frac{a}{L}.$$

Thus if the distance of the code is at least d, the maximum agreement of the code is at most $n-d$. Thus if each s_j is in the code, we must have that:

$$\begin{aligned}\frac{a^2}{\ell n} - \frac{a}{L} \leq & n-d \\ \frac{a^2}{\ell n} - (n-d) \leq & \frac{a}{L}.\end{aligned}$$

If we also have that $a > \sqrt{\ell n(n-d)c}$ and $c > 1$, then we know that $\frac{a^2}{\ell n} - (n-d) > 0$ and $n - d < \frac{a^2}{\ell n c}$. Thus

$$\begin{aligned}L \leq & \frac{a}{\frac{a^2}{\ell n} - (n-d)} \\ \leq & \frac{a\ell n}{a^2 - \ell n(n-d)} \\ \leq & \frac{a\ell n}{a^2 - \ell n \frac{a^2}{\ell n c}} \\ \leq & \frac{\ell n}{a(1-1/c)} \\ = & \ell \frac{n}{a} \frac{1}{1-1/c}.\end{aligned}$$

□

A straightforward corollary of this result is that if there are very few errors in an ℓ-multi-string w then there are only ℓ codewords close to w.

Corollary 4.11 (List Recovery With Few Errors)**.** *Suppose there is a code with alphabet Σ and distance d whose codewords are length n strings. Then for any a, ℓ such that $a > \frac{3n\ell}{1+3\ell}$ and $a > 2\ell\sqrt{n(n-d)}$, and any $w \in \binom{\Sigma}{\ell}^n$ there are at most ℓ strings in Σ^n that agree with w on more than a locations. That is, C is list recoverable with list recovery distance $n-a$, input list length ℓ, and output length ℓ.*

Proof. Use the prior theorem with $c = 4\ell$. This gives an output list length of $L < \ell + 1$, so the list length is at most ℓ. □

There is a general, black box reduction from any list decodable code to a list recoverable code. This reduction works by a greedy algorithm: split the input ℓ-multi-string w into ℓ different strings and then do list decoding with decoding radius γ/ℓ on each of them. All codewords that are contained in w on γ fraction of inputs will be in the output.

Lemma 4.12 (List Decoding to List Recovery)**.** *Suppose C is a code with alphabet Σ which is list decodable with relative list decoding radius $1-\gamma$ and output length L in time T and space S with q queries. Then for any integer ℓ, the code C is also list recoverable with input list length ℓ, output list length $L\ell$, and relative list recovering radius $1-\ell\gamma$ in time $\ell T + O(\ell q \log(|\Sigma|))$ and space $O(\log(\ell|\Sigma|)) + S$ with ℓq queries.*

Proof. For input ℓ-multi-string w, we split w into ℓ strings $w^1, \ldots, w^\ell$ in any arbitrary way as long as $w \subseteq \bigcup'_{i\in[\ell]} w^i$. Then we run list recovery on each w^i with agreement γ and output the results. Consider any codeword y that is contained in w for $\ell\gamma$ fraction of indices. See that for some $i \in [\ell]$ that y must agree with w^i in γ fraction of locations since the expected fraction of symbols y agrees with a random w^i must be at least γ. Thus the algorithm outputs y. The time and number of queries of this algorithm is just the time to run this algorithm ℓ times plus a small overhead to split up w. The space is just the space to run the list decoder plus the space to remember which w^i we are on and to grab the ith symbol from the list of symbols in w. □

4.4 Code Concatenation

Our approach to constructing a list decodable code will result in a large alphabet. A standard way to reduce the alphabet size of a code is by using code concatenation. Code concatenation just replaces every symbol from the code with a codeword from a smaller code.

Definition 4.13 (Code Concatenation)**.** *Suppose for some alphabets $\Sigma_1, \Sigma_2, \Sigma_3$ and integers k, m, n there are codes $C_1 : \Sigma_1^k \to \Sigma_2^n$ and $C_2 : \Sigma_2 \to \Sigma_3^m$. Then we define the concatenated code $C : \Sigma_1^k \to \Sigma_3^{nm}$ (denoted $C = C_1 \circ C_2$) such that for any $x \in \Sigma_1^k$, $i \in [n]$ and $j \in [m]$ we have*

$$C(x)_{(i-1)m+j} = C_2(C_1(x)_i)_j.$$

Now we show that concatenating two efficient list recoverable codes gives an efficient list recoverable code.

Lemma 4.14 (Concatenated Codes Are Efficient)**.** *Suppose C_1 is a code that is list recoverable with list recovering radius d_1, input list length ℓ' and output list length L in time T_1 and space S_1 with q_1 queries. Suppose C_2 is a code that is list recoverable with list recovering radius d_2, input list length ℓ and output list length ℓ' in time T_2 and space S_2 with q_2 queries. Further suppose that C_2 is also encodable in time T_3 and space S_3.*

Then $C_1 \circ C_2$ is list recoverable with recovering radius $d_1 d_2$, input list length ℓ, output list length L in time $T_1 + q_1 T_2 + nT_3$ and space $S_1 + S_2 + S_3$ with $q_1 q_2$ queries.

Proof. The list recovery algorithm is the obvious one. Run the list recovery algorithm for C_1 and each time it queries a codeword symbol, run the list recovery algorithm for C_2, and when it would print a symbol, run the encoder for C_2. The efficiency is direct.

To show the list recovery succeeds. Suppose that there is an ℓ-multi-string w and a codeword y such that for at most $d_1 d_2$ indexes $k \in [nm]$ do we have that y_k is not contained in w_k. Then in particular, for at most d_1 indexes $i \in [n]$ do we have that for d_2 indexes $j \in [m]$ do we have that for $k = (i-1)m + j$ that y_k is not contained in w. Thus since C_2 has list recovering radius d_2, for at most d_1 locations do we have the list of symbols that C_2 provides to C_1 is missing the corresponding symbol from the codeword y. Thus since C_1 has list recovering radius d_1, we have that the codeword in C_1 corresponding to y is recovered. □

To actually use code concatenation, we need an efficiently computable code that is list recoverable and has small alphabet. Since we are only applying this code on $O(\log(n))$ bit inputs, the efficiency of this code does not matter a lot, it just needs very good list recovery.

To apply code concatenation, we need some codes with constant sized alphabets (see [Gur09]). For any constant ϵ, there is a code that is list decodable from $1 - \epsilon$ fraction of errors with an alphabet that only depends on ϵ [GR22, Theorem 1.1].

Lemma 4.15 (List Decodable Codes With any Agreement and Constant Alphabet)**.** *For any $R \in (0, 1)$ and $\epsilon > 0$, there is an infinite family of error-correcting codes of rate at least R over an alphabet of size $(1/\epsilon)^{O(1/\epsilon)}$ that can be encoded in time $\mathsf{poly}(n/\epsilon)$ and can be list decoded from $(1 - R - \epsilon)$-fraction of errors with list size at most $L = 2^{\mathsf{poly}(1/\epsilon)}$ in time $\mathsf{poly}(Ln)$.*

For actually binary codes, we can use [Gur+02].

Lemma 4.16 (List Decodable Codes With Binary Alphabet)**.** *For any fixed $\epsilon > 0$, there exists a polynomial time constructible code family with a binary alphabet and rate $\Omega(\epsilon^4)$ that can be list decoded from $\frac{1}{2} - \epsilon$ fraction of errors in time $\mathsf{poly}(n 2^{\mathsf{poly}(1/\epsilon)})$. They can also be encoded in time $\mathsf{poly}\left(n 2^{\mathsf{poly}(1/\epsilon)}\right)$.*

4.5 List Recovery For Reed-Solomon Codes

For our explicit codes, we will use alphabets that are finite fields.

Definition 4.17 (Finite Field)**.** *We call an integer p a prime power if there exists a prime number p_0 and positive integer such that $p = p_0^a$. For any prime power p, we denote the field of order p as $\mathbb{F}_p$.*

One of the codes we will consider is the Reed-Muller code. This is the code with codewords that are low degree polynomials over many variables.

Definition 4.18 (Reed-Muller Code)**.** *For a field $\mathbb{F}$, degree deg, and number of variables dim the Reed-Muller code with degree deg for $\mathbb{F}^{\mathsf{dim}}$ is the code whose codewords are polynomials $Q : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$ of degree deg represented by evaluating Q at all points in $\mathbb{F}^{\mathsf{dim}}$. So the alphabet of the degree deg Reed-Muller code for $\mathbb{F}^{\mathsf{dim}}$ is $\mathbb{F}$ and the codeword length is $n = |\mathbb{F}|^{\mathsf{dim}}$.*

If $\mathsf{dim} = 1$, we call the code a Reed-Solomon code.

Guruswami and Sudan give a time efficient algorithm for list decoding (and list recovery) of Reed-Solomon codes [Sud97; GS99]. See "Essential Coding Theory" [GRS23, Chapter 12] for this algorithm. More efficient list decoding algorithms were given in follow up works [FB98; KV03]. We use the fast list recovery algorithm given by Alekhnovich [Ale05], and the output list length comes from an extension to the Johnson bound described in Lemma 4.10.

Lemma 4.19 (List Recovery of Reed-Solomon Codes)**.** *Let $\mathbb{F}$ be a finite field. For degree $\mathsf{deg} < |\mathbb{F}|$, let $C : \mathbb{F}^{\mathsf{deg}+1} \to \mathbb{F}^{|\mathbb{F}|}$ be the Reed-Solomon code over $\mathbb{F}$ with degree at most deg. See that deg is the maximum agreement of any two distinct codewords of C.*

Then for any ℓ, distance d, and $c > 1$ such that $|\mathbb{F}| - d > \sqrt{\mathsf{deg}\ell|\mathbb{F}|c}$, we have that C is list recoverable with input list length ℓ, output list length $L \leq \ell \frac{|\mathbb{F}|}{|\mathbb{F}|-d} \frac{1}{1-1/c}$, and list recovery distance d, in time $|\mathbb{F}|\mathsf{poly}\left(\ell \log(|\mathbb{F}|) \frac{|\mathbb{F}|}{\mathsf{deg}}\right)$.

Remark (Soft Decoding)**.** *This list recovery algorithm is more generally a soft decoding algorithm.*

As a special case of this list recovery algorithm, if the number of errors we need to correct and the input list size is small enough, then the output list length is the same as the input list length. So when doing list recovery in the low error regime, input and output list lengths can be equal.

Corollary 4.20 (List Recovery for Reed-Solomon Codes With Few Errors)**.** *Let $\mathbb{F}$ be a finite field. For degree $\mathsf{deg} < |\mathbb{F}|$, let $C : \mathbb{F}^{\mathsf{deg}+1} \to \mathbb{F}^{|\mathbb{F}|}$ be the Reed-Solomon code over $\mathbb{F}$ with degree at most deg. Let ℓ be an integer such that $1 \leq \ell \leq \sqrt{\frac{|\mathbb{F}|}{16\mathsf{deg}}}$.*

Then for any ℓ, we have that C is list recoverable with input list length ℓ, output list length ℓ and list recovery distance $d = \frac{|\mathbb{F}|}{4\ell}$ in time $|\mathbb{F}|\mathsf{poly}\left(\ell \log(|\mathbb{F}|) \frac{|\mathbb{F}|}{\mathsf{deg}}\right)$.

Proof. Set $c = 4\ell \geq 1 + 3\ell$. See that $|\mathbb{F}| - d = |\mathbb{F}| - \frac{|\mathbb{F}|}{4\ell} > \frac{|\mathbb{F}|}{2}$ and that

$$\begin{aligned}\sqrt{\mathsf{deg}\ell|\mathbb{F}|c} \leq& 2\ell\sqrt{\mathsf{deg}|\mathbb{F}|} \\ \leq& 2\sqrt{\frac{|\mathbb{F}|}{16\mathsf{deg}}\mathsf{deg}|\mathbb{F}|} \\ \leq& \frac{|\mathbb{F}|}{2}.\end{aligned}$$

Thus $|\mathbb{F}| - d > \sqrt{\mathsf{deg}\ell|\mathbb{F}|c}$. So applying Lemma 4.19, there is an efficient decoder with output length

$$\begin{aligned}L \leq& \ell \frac{|\mathbb{F}|}{|\mathbb{F}| - d} \frac{1}{1 - 1/c} \\ \leq& \ell \frac{1}{1 - 1/4\ell} \left(1 + \frac{1}{3\ell}\right) \\ \leq& \ell \left(1 + \frac{1}{3\ell}\right) \left(1 + \frac{1}{3\ell}\right) \\ <& \ell + 1.\end{aligned}$$

Since ℓ is the largest integer less than $\ell + 1$, the output list length is at most ℓ. □

Reed-Muller codes (and more generally lifted Reed-Solomon codes [GK16]) are a sub-code of Reed-Solomon codes over a larger field, so the list recovery of Reed-Solomon codes also applies to Reed-Muller codes.

Lemma 4.21 (List Recovery of Reed-Muller Codes)**.** *Let $\mathbb{F}$ be a finite field. For degree $\mathsf{deg} < |\mathbb{F}|$, and dimension dim, let C be the Reed-Mull code over $\mathbb{F}^{\mathsf{dim}}$ with degree at most deg. See that $\frac{\mathsf{deg}}{|\mathbb{F}|}$ is the maximum relative agreement of any two distinct codewords of C.*

Then for any ℓ, relative agreement γ, and $c > 1$ such that $\gamma > \sqrt{\frac{\mathsf{deg}\ell c}{|\mathbb{F}|}}$, we have that C is list recoverable with input list length ℓ, output list length $L \leq \ell \frac{1}{\gamma} \frac{1}{1-1/c}$, and list recovery relative distance $1-\gamma$, in time $|\mathbb{F}|^{\mathsf{dim}} \mathsf{poly}\left(\ell \log(|\mathbb{F}|) \frac{|\mathbb{F}|}{\mathsf{deg}}\right)$.

Note that here $\gamma = \frac{n-d}{n}$. For some of our results, we will use γ instead of d as some results are stated more naturally with relative agreement γ instead of distance d.

4.6 Local Testing

One important feature we need in our codes is a space efficient way to test if a word is close to a codeword or not. The standard way to do this is with local testing [GS06]. Local testing of a code is a randomized algorithm that makes few queries to a word such that the test passes for all codewords and fails with high probability for any word that is far from all codewords. In particular, we want the probability of the test failing to be a good approximation of the distance to the nearest codeword.

Definition 4.22 (Locally Testable Codes (For Low Error))**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ is a locally testable code (LTC) with q queries, randomness R, and approximation factor α if there exists a tester V that takes as input randomness $r \in \{0,1\}^R$ and oracle access to a string $w \in \Sigma_2^m$ and outputs a single bit, denoted $V^w(r)$, such that:*

Locality: *For any $r \in \{0,1\}^R$ and $w \in \Sigma_2^n$ we have that $V^w(r)$ can be computed with only q queries to w.*

Approximation: *For any $w \in \Sigma_2^n$ we have*

$$\frac{\Delta(w,C)}{n\alpha} \leq \Pr_{r \in \{0,1\}^R}[V^w(r)] \leq \frac{\alpha \Delta(w,C)}{n}.$$

If the approximation factor α is not specified, it is assumed to be 2.

For non-uniform correction, a general local tester who can approximate the number of errors within any constant multiplicative factor is enough. We call this the low error (high agreement) setting because we only need our local tester to differentiate between low error and constant fraction of error. In the low error regime, for $\alpha = 2$, if the test fails with probability 0.01, then we know the word has relative distance within 0.02 of a codeword. However, if the test fails with probability 0.99, then we still only know the relative distance to a codeword is 0.495. This is the standard setting and there has been extensive work constructing codes that are locally testable in the low error regime. Locally testable codes in the low error regime include multiplicity codes [KTS22], tensor codes [BS06; Vid15; Kop+17], lifted Reed-Solomon codes [GKS13], codes based on certain expander graphs [Din+22; PK22; LH22], codes based on probabilistically checkable proofs [Ben+03], and Reed-Muller codes [PS94; FS95].

On the other hand, we could also ask our local tests to not just give a good estimate of the distance to a code, but also a close estimate of the amount of the agreement with a codeword. We call this the high error (low agreement) setting. In this setting, a word only fails the test with probability 0.99 if it is 0.98 far from every codeword. This setting is studied far less often. A prominent example of a code with tests in the high error regime are Reed-Muller codes [RS97; AS97; BDLN17; MZ23; Har+24]. These works show that for low degree testing (which is a local test for Reed-Muller codes) the subspace tests give the distance to low degree codewords up to a very small additive error. Other codes with tests in the high error regime include the direct product code [DG08].

For our results, we need local testers in the *high error* regime that are very randomness efficient. In the low error regime, there is an efficient local test based on lines in an ϵ biased set [Ben+03]. In the high

error regime, there is an efficient local test based on subspaces with directions from a subfield [MR06]. See Lemma 6.3.

4.7 Locally Correctable Codes

To construct space efficient global correctors, we will use local correction. A locally correctable code is a code with a local corrector. A local corrector is a randomized algorithm, D, that when given oracle access to a string w that is a slightly corrupted version of a codeword y, the corrector D can compute any symbol of y with high probability using few queries to w. See [Yek12] for an overview of locally correctable and locally decodable codes.

Definition 4.23 (Locally Correctable Codes (LCC))**.** *For any code $C : \Sigma_1^k \to \Sigma_2^n$ with distance greater than $2d$, we say that C is a locally correctable code (LCC) if there exists some local corrector D that takes as input an index $i \in [m]$, a randomness string $r \in \{0,1\}^R$, and oracle access to a string $w \in \Sigma_2^n$ and outputs a single symbol, denoted $D^w(i,r)$, such that*

Locality: *On any input string $w \in \Sigma_2^n$, index $i \in [n]$, and randomness $r \in \{0,1\}^R$ we have that $D^w(i,r)$ can be computed with only q queries to w.*

Soundness: *There is an $s < \frac{1}{2}$ such that for any $w \in \Sigma_2^n$ and $x \in \Sigma_1^k$ with $\Delta(w, C(x)) \leq d$ and any $i \in [n]$ we have*

$$\Pr_{r \in \{0,1\}^R}[D^w(i,r) \neq C(x)_i] \leq s.$$

We call d the correcting radius, q the number of queries, and s the soundness of D. If soundness s is not specified, it is assumed to be $\frac{1}{3}$. If correcting radius d is not specified, it is assumed to be $\Omega(n)$.

Many constructions of locally decodable and correctable codes have been given [KSY14; HOW13; GKS13; Kop+17].

To do list correction, we can't start with just a unique local correction algorithm. We need a local list correction algorithm. This is a *randomized* local algorithm that makes few queries to the input, and then outputs a few local decoders such that each nearby codeword is decoded by one of the local decoders.

Definition 4.24 (Locally List Correctable Codes (LLCC))**.** *Take any code $C : \Sigma_1^k \to \Sigma_2^n$ and suppose that there exists a randomized algorithm D with oracle access to an input word $w \in \Sigma_2^n$ and outputs a collection of randomized local correctors $D_1, \ldots, D_L$. For each $j \in [L]$, the local corrector D_j takes as input an index $i \in [n]$, a randomness string $r \in \{0,1\}^R$, and oracle access to a string $w \in \Sigma_2^n$ and outputs a single symbol, denoted $D_j^w(i,r)$.*

We say that D is a local list correcting algorithm for C if

Locality: *For some q, we have that D only makes q queries to w and each D_j also only makes q queries to w.*

Soundness: *There are $s_1, s_2 \in (0,1)$ and distance d such that for any $w \in \Sigma_2^n$ with probability at least $1 - s_1$, D outputs $D_1, \ldots D_L$ such that for every and $x \in \Sigma_1^k$ with $\Delta(w, C(x)) \leq d$, we have that for some $j \in [L]$ that*

$$\Pr_{r \in \{0,1\}^R}[D_j^w(i,r) \neq C(x)_i] \leq s_2.$$

We call d the list correcting radius, q the number of list correcting queries, and L the list length. We call s_1 the list choosing soundness and s_2 the list correcting soundness. If the soundness s_1 or s_2 is not specified, then they are assumed to be $\frac{1}{3}$. If the list length is not specified, it is assumed to be $O(1)$.

Many of the local correcting results extend to local list decoding [GK16; HRZW17; Gop+18; Kop+23]. Just as one can generalize list decoding to list recovery, we also can generalize local list decoding to local list recovery.

Definition 4.25 (Locally List Recoverable Codes (LLRC))**.** *Take any code $C : \Sigma_1^k \to \Sigma_2^n$ and suppose that there exists a randomized algorithm D with oracle access to multi-string $w \in \binom{\Sigma_2}{\ell}^n$ and outputs a collection of randomized local correctors $D_1, \ldots, D_L$. For each $j \in [L]$, the local corrector D_j takes as input an index $i \in [n]$, a randomness string $r \in \{0,1\}^R$, and oracle access to a multi-string $w \in \binom{\Sigma_2}{\ell}^n$ and outputs a single symbol, denoted $D_j^w(i, r)$.*

We say that D is a local list recovery algorithm for C if

Locality: *For some q, we have that D only makes q queries to w and each D_j also only makes q queries to w.*

Soundness: *There are $s_1, s_2 \in (0, 1)$ and distance d such that for any $w \in \binom{\Sigma_2}{\ell}^n$ with probability at least $1 - s_1$, D outputs $D_1, \ldots D_L$ such that for every and $x \in \Sigma_1^k$ with $|\{i : C(x)_i \notin w_i\}| \leq d$, we have that for some $j \in [L]$ that*

$$\Pr_{r \in \{0,1\}^R}[D_j^w(i, r) \neq C(x)_i] \leq s_2.$$

We call d the list recovring radius, q the number of list recovery queries, ℓ the input list length and L the output list length. We call s_1 the list choosing soundness and s_2 the list correcting soundness. If the soundness s_1 or s_2 is not specified, then they are assumed to be $\frac{1}{3}$. If the list length is not specified, it is assumed to be $O(1)$.

To get our decoders for locally list recoverable codes, we will further need that they are both locally testable and have an efficient global unique decoder. A prior result by Cook and Moshkovitz [CM25] showed that all *typical* locally correctable codes have efficient unique decoders. So we further define a typical locally correctable code to be one that is systematic, smooth, and has perfect completeness.

A systematic code is a code such that any message is contained (as plain text) in the associated codeword. We use systematic codes since it gives a straightforward way to get the message from an uncorrupted codeword. Similar results hold for any code with an efficient way to recover the message from an uncorrupted codeword. Systematic codes provide a simple mechanism for doing this.

Definition 4.26 (Systematic Code)**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ is systematic if for all $i \in [k]$ there exists $j \in [j]$ such that for all $x \in \Sigma_1^k$ we have that $x_i = C(x)_j$.*

A smooth code is a code with a local corrector such that when it corrects any particular symbol, the corrector queries every index of the input with about equal probability. In particular, no index is queried more often than about $\frac{2q}{n}$ times in expectation.

Definition 4.27 (Smooth)**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ with a local corrector D (where the randomness of D is R) is β smooth if for all $w \in \Sigma_2^n$ and $i, j \in [n]$:*

$$\Pr_{r \in \{0,1\}^R}[D^w(i, r) \text{ queries index } j \text{ of } w] \leq \frac{\beta}{n}.$$

If β is unspecified and D is q query, we assume $\beta = 2q$.

A non-adaptive code is a code with a local corrector such that the indexes of the input which are queried *only* depends on the randomness and the index of the symbol being decoded, *not* on the input being corrected. Put another way, given the index to decode and a random string, the corrector can specify each of the q locations it will query before querying them.

Definition 4.28 (Non-Adaptive)**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ with a local corrector D (where the randomness of D is R) is non-adaptive if for all $i \in [n]$ and $r \in \Sigma_2^n$, we have that $D^w(i, r)$ always queries the same indexes in w for any $w \in \Sigma_2^n$.*

Finally, a code with a local corrector is said to have perfect completeness if, when it is given a valid codeword, it always outputs the symbols from that codeword. Put another way, local corrections of an uncorrupted codeword always output that codeword.

Definition 4.29 (Perfect Completeness)**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ with a local corrector D (where the randomness of D is R) has perfect completeness if for all $x \in \Sigma_1^k$, for any $i \in [n]$ and $r \in \{0,1\}^R$ we have that*

$$D^{C(x)}(i, r) = C(x)_i.$$

Now we define a typical LCC to be any systematic code with a corrector that is smooth, non-adaptive and has perfect completeness. These are standard properties of locally correctable codes, and all the standard constructions of LCC are typical (with only minor changes).

Definition 4.30 (Typical Locally Correctable Codes)**.** *For any systematic code $C : \Sigma_1^k \to \Sigma_2^n$ we say that C is a typical LCC if it is an LCC and has a corrector that is smooth, non-adaptive, and has perfect completeness.*

5 Low Error List Recovery

Reed-Muller codes are already well known to be locally list recoverable, and locally testable. So one could use the results in Appendix A.1 to give *non-uniform* efficient list recovery algorithms for Reed-Muller codes. But we can give a *uniform* list decoding algorithm for Reed-Muller codes.

We start by giving a list recovery algorithm for the case where there is low error. List recovery with very few errors is very similar to the unique decoding case of [CM25] except that instead of improving a string to a nearby codeword, we are improving a multi-string. In Section 6 we will show how to locally reduce from a high error string to a multi-string with very low error. Combining these two results will give our list decoding algorithm for Reed-Muller codes.

5.1 List Improving Sets

An improving set is a set of deterministic local functions that takes as input a string near a codeword and outputs another string that in expectation is closer to that codeword. Improving sets were defined in [CM25] to allow time and space efficient global decoding. In this paper we use a generalization called a list improving set which takes as input a multi-string that is near a codeword multi-string (a multi-string that is exactly made from codewords) and in expectation outputs a multi-string closer to that codeword multi-string.

One technical note is that we separate the kind of errors in an input multi-string into two kinds. The in-codeword, or completeness, errors where there are symbols missing in the multi-string that are in a codeword, and the out-of-codeword, or soundness, errors where there are symbols in the multi-string that are not in a codeword. By separating the two we are able to handle many more out-of-codeword errors since we start with fewer in-codeword errors.

Definition 5.1 (List Improving Set)**.** *Take any alphabet Σ, integers n, ℓ, and a code C whose codewords are in Σ^n. Let L be an integer and $\mathcal{I}$ be a set of deterministic functions from ℓ-multi-strings to L-multi-strings. Then for any integer d and numbers $\delta, \alpha > 0$, we say $\mathcal{I}$ is a below d factor δ list improving set with out-of-codeword tolerance $1 - \alpha$, input list length ℓ and output list length L for C if*

Completeness: *For any ℓ-multi-string w, and codeword multi-string Q where*

$$\Pr_{i \in [n]} [Q_i \not\subseteq w_i] \leq \frac{d}{n}$$

we have that

$$\Pr_{f \in \mathcal{I}, i \in [n]} [Q_i \not\subseteq f(w)_i] \leq \delta \Pr_{i \in [n]} [Q_i \not\subseteq w_i].$$

Soundness: *For any ℓ-multi-string w, and ℓ-codeword multi-string Q with*

$$\Pr_{i \in [n]} [w_i \not\subseteq Q_i] \leq 1 - \alpha$$

we have that

$$\Pr_{f \in \mathcal{I}, i \in [n]} [f(w)_i \not\subseteq Q_i] \leq \delta \Pr_{f \in \mathcal{I}, i \in [n]} [w_i \not\subseteq Q_i].$$

Efficient: *We say that $\mathcal{I}$ is q query, time T, and space S if for each function in $f \in \mathcal{I}$ and $i \in [n]$, we can compute $f(w)_i$ using only q queries to w in time T and space S.*

A straightforward consequence of this definition is a test for how close a multi-string is to a codeword multi-string. Specifically, we can test if some y is closer to the codewords that w is close to.

Lemma 5.2 (List Improving Set to Tester)**.** *Take any alphabet Σ, integers n, ℓ, and a code C whose codewords are in Σ^n. Suppose that $\mathcal{I}$ is a q query, time T, space S, below d factor δ list improving set with out-of-codeword tolerance $1-\alpha$, input list length ℓ and output list length L for C.*

Then there is an algorithm V that takes as input ℓ-multi-string w and L-multi-string y and outputs a number $\beta \in [0,1]$ such that if for some ℓ-codeword multi-string Q we have

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \frac{d}{n}$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1-\alpha$$

then

$$\left|\beta - \Pr_{i\in[n]}[Q_i \neq y_i]\right| \leq 2\delta \Pr_{i\in[n]}[Q_i \neq w_i]$$

and V runs with $nq|\mathcal{I}|+n$ queries in time $O(nT|\mathcal{I}|)$, and space $S+\log(n|\mathcal{I}||\Sigma|))$.

Proof. The idea is that we will run every function $f \in \mathcal{I}$ on w to get a good estimate of Q and compare that to y. See that by the completeness and soundness of $\mathcal{I}$ we have that

$$\begin{aligned}\Pr_{f\in\mathcal{I},i\in[n]}[Q_i \neq f(w)_i] \leq& \Pr_{f\in\mathcal{I},i\in[n]}[Q_i \not\subseteq f(w)_i] + \Pr_{f\in\mathcal{I},i\in[n]}[f(w)_i \not\subseteq Q_i]\\ \leq& \delta \Pr_{i\in[n]}[Q_i \not\subseteq w_i] + \delta \Pr_{f\in\mathcal{I},i\in[n]}[w_i \not\subseteq Q_i]\\ \leq& 2\delta \Pr_{i\in[n]}[Q_i \neq w_i].\end{aligned}$$

So by a reverse triangle inequality, we have

$$\left|\Pr_{f\in\mathcal{I},i\in[n]}[Q_i \neq y_i] - \Pr_{f\in\mathcal{I},i\in[n]}[y_i \neq f(w)_i]\right| \leq 2\delta \Pr_{i\in[n]}[Q_i \neq w_i].$$

The algorithm outputs $\beta = \Pr_{f\in\mathcal{I},i\in[n]}[y_i \neq f(w)_i]$ and our result holds. □

Now if we run this test with y being the output of different functions in the list improving set, then we can find a function in the improving set that improves the multi-string significantly.

Lemma 5.3 (List Improving Set Gives Partial Codewords List Recovery)**.** *Take any alphabet Σ, integers n, ℓ, and a code C whose codewords are in Σ^n. Suppose that $\mathcal{I}$ is a q query, time T, space S, below d factor δ list improving set with out-of-codeword tolerance $1-\alpha$, input list length ℓ and output list length L for C. Then there is an algorithm D that takes as input an ℓ-multi-string w and outputs the index of some $f \in \mathcal{I}$ such that if for some ℓ-codeword multi-string Q we have*

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \frac{d}{n}$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1-\alpha$$

then we also have that

$$\Pr_{i\in[n]}[f(w)_i \neq Q_i] \leq 6\delta \Pr_{i\in[n]}[w_i \neq Q_i].$$

The algorithm D runs with $nq|\mathcal{I}|^2$ queries, time $O(nT|\mathcal{I}|^2)$, and space $S+O(\log(n|\mathcal{I}||\Sigma|))$.

Proof. The idea is to run the local test from Lemma 5.2 with $y = f(w)$ for each $f \in \mathcal{I}$ and output the function f that outputs the smallest β. All that we need to show is that there is some $f \in \mathcal{I}$ that will fail with probability at most $4\delta \Pr_{i\in[n]}[w_i \neq Q_i]$. Recall the local tester in Lemma 5.2 chooses a random $f' \in \mathcal{I}$ and compares $f(w)$ to $f'(w)$. See that

$$\begin{aligned}\Pr_{f'\in\mathcal{I},f\in\mathcal{I},i\in[n]}[f(w)_i \neq f'(w)_i] \leq& 2 \Pr_{f\in\mathcal{I},i\in[n]}[Q_i \neq f(w)_i] \\ \leq& 4\delta \Pr_{i\in[n]}[w_i \neq Q_i].\end{aligned}$$

Thus since in expectation a random $f \in \mathcal{I}$ fails with probability at most $4\delta \Pr_{i\in[n]}[w_i \neq Q_i]$, some f achieves this expectation. That is, for some f_i, the test of Lemma 5.2 outputs a $\beta \leq 4\delta \Pr_{i\in[n]}[w_i \neq Q_i]$. For such an f, we have that

$$\Pr_{i\in[n]}[f(w)_i \neq Q_i] \leq 6\delta \Pr_{i\in[n]}[w_i \neq Q_i].$$

So the protocol outputs such an f. □

Using this, we can get codewords list recovery from a list improving set. This is list recovery where instead of actually outputting the codewords, we output a multi-string that exactly contains the codewords.

Lemma 5.4 (List Improving Set Gives Codewords List Recovery)**.** *Take any alphabet Σ, integers n, ℓ, and a code C whose codewords are in Σ^n. Suppose that $\mathcal{I}$ is a $q \geq 2$ query, time T, space S, below d factor $\delta \leq \frac{1}{12}$ list improving set with out-of-codeword tolerance $1-\alpha$, input list length ℓ and output list length L for C. Then there is an algorithm D that takes as input an ℓ-multi-string w and outputs the description of a function f such that if for some ℓ-codeword multi-string Q we have that*

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \frac{d}{n}$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1-\alpha$$

then the function f has

$$f(w) = Q.$$

Let $k = \lceil \log(n+1)/\log(1/6\delta) \rceil$. Then the algorithm f has description length $O(k \log(|\mathbb{F}|))$, and each output location can be computed with q^k queries in time $O(Tq^k)$ and space $O(kS)$. The algorithm D runs with $O(nq^k|\mathcal{I}|^2)$ queries in time $O(nTq^k|\mathcal{I}|^2)$, and space $O(kS + \log(n|\mathcal{I}||\Sigma|))$.

Proof. The idea is to run Lemma 5.3 recursively k times. Each time it will decrease the fraction of errors by a factor of 6δ. After k rounds, this will decrease the fraction of error down to $(6\delta)^k \leq \frac{1}{n+1}$, which would require no errors.

For a more formal induction, suppose that we have an algorithm D_i that runs with $O(nq^i|\mathcal{I}|^2)$ queries in time $O(nTq^i|\mathcal{I}|^2)$, and space $O(iS + \log(n|\mathcal{I}||\Sigma|))$ and outputs a function f_i that has description length $O(i \log(|\mathbb{F}|))$, and each output location can be computed with q^i queries in time $O(Tq^i)$ and space $O(iS)$ such that $\Pr_{i\in[n]}[f_i(w) = Q_i] \leq (6\delta)^i$. For $i = 1$, we get this by using Lemma 5.3 directly. In the general case, we get D_{i+1} by running Lemma 5.3 on $f_i(w)$ and then outputting f_{i+1} as the found f composed with f_i. This only increases the time and number of queries of D_{i+1} and f_{i+1} by a factor of q. Since $q > 2$, this is a geometric sequence, so the time of running all $D_1, \ldots, D_i$ are factored into the O. Similarly space only increases by an additive amount S since there is another level in the recursion. □

5.2 Correcting Lines

To run our list recovery algorithm, we need to first construct list improving sets. In fact, our list improving sets use the same correcting lines used by [CM25] to construct their improving sets. Our corrector uses a set of pseudorandom lines through a point, corrects along each, and takes a majority vote. We will now more formally define what properties our lines must have to give us a good list improving set.

The first concept is called "correcting lines". Correcting lines for a given index $x \in \mathbb{F}^{\mathsf{dim}}$ are just a set of lines that pass through x. A set of such correcting lines will be good if less than half over-sample the set of corruptions.

Definition 5.5 (Correcting Lines for x)**.** *Let dim, c be natural numbers, take $\epsilon > 0$, take $\mathbb{F}$ to be a field, take $x \in \mathbb{F}^{\mathsf{dim}}$. $\mathcal{L}_x$ forms c* correcting lines *for x if $\mathcal{L}_x = (l_1, \ldots, l_c)$, where $l_1, \ldots, l_c$ are lines in $\mathbb{F}^{\mathsf{dim}}$ such that for all $i \in [c]$ we have that $l_i(0) = x$.*

For any set $A \subseteq \mathbb{F}^{\mathsf{dim}}$, let $\mu = \frac{|A|}{|\mathbb{F}^{\mathsf{dim}}|}$. We say that $\mathcal{L}_x$ is ϵ-sampling for A if

$$\Pr_{i \in [c]}\left[\left|\Pr_{\lambda \in \mathbb{F}}[l_i(\lambda) \in A] - \mu\right| \geq \epsilon\right] < 1/2.$$

A correcting lines set is a set of correcting lines (so it is a set of sets of lines) such that for any set A, most correcting lines are ϵ-sampling for A. If this is true, then we can do error correction. Specifically, we need the probability that correcting lines are not ϵ-sampling to be proportional to the size of A so the correction always decreases the number of errors.

Definition 5.6 (Correcting Lines Set)**.** *Let dim, c, k be natural numbers and take $\mathbb{F}$ to be a field. A* correcting lines set *is a family $\mathcal{L} = \{\mathcal{L}_{x,i} : x \in \mathbb{F}^{\mathsf{dim}}, i \in [k]\}$, where each $\mathcal{L}_{x,i}$ forms c correcting lines for x. We say $\mathcal{L}$ forms a below d correcting lines set with strong confidence error δ, and accuracy error ϵ if for all $A \subseteq \mathbb{F}^{\mathsf{dim}}$ with $|A| \leq d$, we have that*

$$\Pr_{x \in \mathbb{F}^{\mathsf{dim}}, i \in [k]}[\mathcal{L}_{x,i} \text{ is not } \epsilon\text{-sampling for } A] \leq \delta \min\{\mu, 1 - \mu\}$$

where $\mu = \frac{|A|}{|\mathbb{F}^{\mathsf{dim}}|}$

We say $\mathcal{L}$ is time T and space S uniform if for each $x \in \mathbb{F}^{\mathsf{dim}}$ and $i \in [k]$ we have that every $l \in \mathcal{L}_{x,i}$ can be computed in time T and space S. We call c the line count of $\mathcal{L}$.

We use the same correcting lines as used in the prior work. The following result is proved during the proof of [CM25, Lemma 8.11]. This correcting line set is made by using an ϵ-biased set to make a subspace sampler, and then sampling lines in that subspace.

Lemma 5.7 (Good Correcting Lines Exist)**.** *Take any field $\mathbb{F}$ with $|\mathbb{F}| \geq 101$, any dimensions dim and a, and $\epsilon > 0$. Then there exists $\mathcal{L}$ that is a below $\epsilon|\mathbb{F}|^{a\mathsf{dim}}/12$ correcting lines set for $\mathbb{F}^{a\mathsf{dim}}$ with accuracy error $11\epsilon/12$,* strong *confidence error*

$$O\left(\frac{a^{2a+1}400^a}{|\mathbb{F}|^a\epsilon^2}\right),$$

and line count $|\mathbb{F}|$. Further the size of the final correcting lines set is

$$|\mathcal{L}| = |\mathbb{F}|^{a(\mathsf{dim}+2a+O(1))}\mathsf{poly}(\mathsf{dim})$$

and $\mathcal{L}$ is time and space $\mathsf{poly}(|\mathbb{F}|^a\mathsf{dim})$ uniform.

In prior work, Cook and Moshkovitz [CM25] showed that correcting line sets give improving sets for Reed-Muller codes. We now show that they further give list improving sets.

Lemma 5.8 (Improving Set from a Correcting Lines Set)**.** *Suppose that for some dimension dim and field $\mathbb{F}$ we have $\mathcal{L}$ that is a time T space S uniform, below d correcting lines set for $\mathbb{F}^{\mathsf{dim}}$ with size $|\mathcal{L}|$, strong confidence error δ, accuracy error ϵ, and line count c. Let $n = |\mathbb{F}^{\mathsf{dim}}|$.*

Let C be the Reed-Muller code for field $\mathbb{F}$, dimension dim and degree deg. Suppose for some integer ℓ, and some $\gamma \in (0,1)$ we have that $1 - \frac{d}{n} - \epsilon > \gamma > \sqrt{\frac{1.001\mathsf{deg}\ell}{|\mathbb{F}|}}$.

Then there is a $c(|\mathbb{F}|-1)+1$ query, time $c\left(T + O\left(|\mathbb{F}|\left(\mathsf{dimpolylog}(|\mathbb{F}|) + \mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)\right)\right)\right)$ space $S + O((\mathsf{dim}+c)\log(|\mathbb{F}|)) + |\mathbb{F}|\mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$ below d, factor δ list improving set for C with size $|\mathcal{L}|/|\mathbb{F}^{\mathsf{dim}}|$, out-of-codeword tolerance $\gamma - \epsilon - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}$, and input list length ℓ. Further if $\gamma \geq 1 - \frac{1}{4\ell}$ and $\ell \leq \sqrt{\frac{3|\mathbb{F}|}{20\mathsf{deg}}}$, then the output list length is also ℓ.

Proof. Let $k = |\mathcal{L}|/n$. We now describe for $i \in [k]$, how the ith function in the improving outputs the symbols at index $x \in \mathbb{F}^{\mathsf{dim}}$. For every line $l \in \mathcal{L}_{x,i}$, it queries everywhere in that line and then runs list recovery on that line with agreement γ using Lemma 4.21[1]. Specifically, the local recovery of the polynomial restricted to a single line outputs a list of length $L \leq \ell \frac{1}{\gamma} \frac{1}{1-1/c'}$ (for any c' where $\gamma > \sqrt{\frac{c'\mathsf{deg}\ell}{|\mathbb{F}|}}$) that contains every polynomial that agrees with that line on a γ fraction of inputs and runs in time $|\mathbb{F}|^{\mathsf{dim}}\mathsf{poly}\left(\ell \log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$. Finally, our function outputs a symbol σ if for the majority of lines $\mathcal{L}_{x,i}$ the list recovery algorithm outputs a polynomial that evaluates to σ at x.

Now we verify it has the required properties.

Completeness: Take any ℓ-multi-string w and codeword multi-string Q where

$$\Pr_{x \in \mathbb{F}^{\mathsf{dim}}}[Q_x \not\subseteq w_x] \leq \frac{d}{n}.$$

Let A be the set of locations x where $Q_x \not\subseteq w_x$. Since $\mathcal{L}$ is a correcting line set, we know that the probability over $x \in \mathbb{F}^{\mathsf{dim}}$ and $i \in [k]$ that $\mathcal{L}_{x,i}$ is not ϵ-sampling for A is at most $\delta \Pr_{x \in \mathbb{F}^{\mathsf{dim}}}[Q_x \not\subseteq w_x]$. Now we just need to show that if $\mathcal{L}_{x,i}$ is ϵ-sampling for A, then the local corrector will output all the symbols in Q_x.

If $\mathcal{L}_{x,i}$ is ϵ-sampling for A, this means that for less than $\frac{1}{2}$ fraction of lines in $\mathcal{L}_{x,i}$ do we intersect with A more than $\frac{|A|}{n} + \epsilon \leq \frac{d}{n} + \epsilon$ fraction of points. Thus for more than half the lines in $\mathcal{L}_{x,i}$ we sample points j where $Q_j \subseteq w_j$ more than $1 - \frac{d}{n} - \epsilon > \gamma$ fraction of the time. Since most of the lines agree with Q on a γ fraction of locations, the local list recovery algorithm outputs every symbol in Q_x.

Soundness: Take any ℓ-multi-string w and ℓ-codeword multi-string Q with

$$\Pr_{x \in \mathbb{F}^{\mathsf{dim}}}[w_x \not\subseteq Q_x] \leq \gamma - \epsilon - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}.$$

Similar to the completeness case, let A be the set of locations x such that $w_x \not\subseteq Q_x$. We only need to show that if $\mathcal{L}_{x,i}$ is ϵ-sampling for A, then we will not output any symbols outside Q. If $\mathcal{L}_{x,i}$ is ϵ-sampling for A, then we have that at least half of the lines have less than $\gamma - \epsilon - \frac{\ell\mathsf{deg}}{|\mathbb{F}|} + \epsilon = \gamma - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}$ fraction of indexes in A. Since there are at most ℓ polynomials in Q and any other polynomial can only agree with any of these ℓ polynomials at $\frac{\mathsf{deg}}{|\mathbb{F}|}$ places, at least half the lines can only agree with any polynomial outside Q on less than $\gamma - \frac{\ell\mathsf{deg}}{|\mathbb{F}|} + \ell\frac{\mathsf{deg}}{|\mathbb{F}|} = \gamma$ fraction of the points. So such lines will not correct to any other polynomials. Thus if $\mathcal{L}_{x,i}$ is ϵ-sampling for A it will not output any symbols outside Q_x

Efficient: Finally, each local function only needs to query the c lines (which all intersect at a point), and only requires the time to find the c lines in $\mathcal{L}_{x,i}$, which is $O(cT)$, plus for each of the c lines the time to compute the points in that line, $O(\mathsf{polylog}(|\mathbb{F}|)|\mathbb{F}|\mathsf{dim})$, plus time $|\mathbb{F}|\mathsf{poly}\left(\ell \log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$ to run the local list recovery on that line. Altogether this gives the claimed time.

For space, we only need the space to to find the lines, S, plus the space to store a description of the line, $\mathsf{dim}\log(|\mathbb{F}|)$, plus which line we are on, $O(\log(c))$, plus the space to run list recovery on a line, $|\mathbb{F}|\mathsf{poly}\left(\ell \log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$, plus the space to store how many times each symbol has been decoded by a line, which is at most $O(|\mathbb{F}|\log(cL))$ where L is the output list length, which is at most $|\mathbb{F}|\mathsf{poly}\left(\ell \log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$. We note that a function in the improving set computes each coordinate independently.

When $\gamma \geq 1 - \frac{1}{4\ell}$ and $\ell \leq \sqrt{\frac{3|\mathbb{F}|}{20\mathsf{deg}}}$, the output list length is because the agreement is so high, similar to Corollary 4.20.

[1]We note that we could have used Corollary 4.20 since a Reed-Muller code restricted to a line is a Reed-Solomon code. We only use Lemma 4.21 since it is stated in terms of γ which is more convenient for us.

In more detail, first let $c' = 3.5\ell$. See that $\gamma \geq 1 - \frac{1}{c'}$ and

$$\begin{aligned}\sqrt{\frac{c'\mathsf{deg}\ell}{|\mathbb{F}|}} =& \ell\sqrt{3.5\frac{\mathsf{deg}}{|\mathbb{F}|}}\\ \leq& \sqrt{\frac{10.5}{20}}\\ <& 0.73\\ <& 1 - \frac{1}{4\ell}\\ \leq& \gamma.\end{aligned}$$

See that $c' \geq 1 + 2.5\ell$. Thus the output list length is

$$\begin{aligned}L \leq& \ell\frac{1}{\gamma}\frac{1}{1-1/c'}\\ \leq& \ell\left(\frac{1}{1-1/c'}\right)^2\\ \leq& \ell\left(\frac{1}{1-1/(1+2.5\ell)}\right)^2\\ \leq& \ell\left(1+\frac{1}{2.5\ell}\right)^2\\ <& \ell + 1.\end{aligned}$$

Thus the output list has length at most ℓ.

□

5.3 Codewords Recovery for Reed-Muller Codes to List Recovery

Now using the correcting lines, we can get list improving sets, which give us codewords list recovery. Now we just need to convert the codewords list recovery into regular list recovery. Essentially, we need some way to associate each symbol in the multi-string with the specific codeword its associated with so we can print the codewords in order.

We use the special properties of Reed-Muller codes to show how to take a multi-string codeword and efficiently output the corresponding list of codewords. In particular, that for every two points, there is a local code that contains those two points (namely, the low degree polynomials on the line between them).

Lemma 5.9 (Efficient List From Codeword Multi-String Codeword)**.** *Take integers $\mathsf{dim}, \mathsf{deg}, \ell$ and field $\mathbb{F}$ such that $\ell < \sqrt{\frac{|\mathbb{F}|}{\mathsf{deg}}}$. Suppose that $Q'^1, \ldots, Q'^\ell$ are elements of the degree deg Reed-Muller code over $\mathbb{F}^{\mathsf{dim}}$. Let Q be the codeword multi-string $Q = \bigcup'_{j\in[\ell]} Q'^j$.*

Then there is an $O(|\mathbb{F}|^{\mathsf{dim}})$ query algorithm running in time $O(|\mathbb{F}^{\mathsf{dim}}|\ell)$ and space $O(\ell + \log(|\mathbb{F}^{\mathsf{dim}}|))$ that outputs a length $\lceil(\ell + \mathsf{dim})\log(|\mathbb{F}|)\rceil$ description of a list of ℓ functions $f_1, \ldots, f_\ell$ such that (under some ordering of the functions) for each $j \in [\ell]$, $f_j(Q) = Q^j$ and each f_j can compute an index of its output in $|\mathbb{F}|$ queries and time $|\mathbb{F}|\mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$.

Proof. The algorithm is simple: search Q until we find the index, $x \in \mathbb{F}^{\mathsf{dim}}$, with ℓ symbols in it. There will be such a coordinate because each pair of codewords can only agree on $\frac{\mathsf{deg}}{|\mathbb{F}|}$ fraction of places, thus at only at most $\ell^2\frac{\mathsf{deg}}{|\mathbb{F}|} < 1$ fraction of places can any two of the codewords agree. Once x is found, each symbol in Q_x uniquely identifies which codeword it corresponds to. Each f_j remembers x and one symbol $\sigma_j \in Q_x$. Finally, for each f_j to decode a symbol x', it looks at the line through x and x' and runs Reed-Solomon list recovery on this line to get a list of polynomials that are the codewords $Q^1, \ldots, Q^j$ restricted to this line. Only Q^j restricted to this line outputs σ_j at x, so use that Q^j restricted to this line to output the symobl at x'. □

5.4 Low Error List Recovery For Reed-Muller Codes

Now, we can use our correcting lines from Lemma 5.7 with Lemma 5.8 to get a list improving set. We can use that list improving set with Lemma 5.4 to recover the codeword multi-string. Finally we use Lemma 5.9 to turn that codeword multi-string into the distinct codewords. All together, this gives us efficient list *recovery* in the *low error* regime. In Section 7 we combine Lemma 5.10 with the results in Section 6 to get list decoding in the *high error* regime.

Lemma 5.10 (Low Error List Recovery for Reed-Muller)**.** *There is some large constant c_0 such that the following holds. Take any field $\mathbb{F}$ with $|\mathbb{F}| \geq 101$, any dimensions dim and a, integer ℓ, and $\alpha > 0$. Let $n = |\mathbb{F}^{a\mathsf{dim}}|$. Let C be the Reed-Muller code for $\mathbb{F}^{a\cdot\mathsf{dim}}$ and degree deg. Suppose for some integer ℓ, each of the following holds:*

1. $\log(|\mathbb{F}|) \geq c_0(\log(a) + \log(1/\alpha))$
2. $\alpha < \frac{1}{100\ell}$
3. $\frac{\ell\mathsf{deg}}{|\mathbb{F}|} < 15\alpha$

Then there is an algorithm D that takes as input an ℓ-multi-string w and outputs the description of ℓ functions $f_1, \ldots, f_\ell$ such that the following holds. Suppose for some codewords $Q'^1, \ldots, Q'^j$ we have codeword multi-string $Q = \bigcup'_{j\in[\ell]} Q'^j$. Then if

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \alpha$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1 - 50\alpha$$

then for each $j \in [\ell]$ there is a $j' \in [\ell]$ such that $f_{j'}(w) = Q'^j$.

Further, D runs with

$$n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$$

queries in time

$$n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}\ell),$$

and space

$$\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell).$$

And each f_i can compute a single index of its output with $n^{4/a}\mathsf{poly}(|\mathbb{F}|)$ queries, time $n^{4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$ and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$.

Proof. First we choose parameters for Lemma 5.7 and Lemma 5.8. Set $\epsilon = 12\alpha$. Set $\gamma = 1 - 2\epsilon$. See that $\gamma = 1 - 24\alpha > 1 - \frac{1}{4\ell}$. See that

$$\begin{aligned}\frac{\ell\mathsf{deg}}{|\mathbb{F}|} \leq& 15\alpha\\ \leq& \frac{15}{100\ell}\\ \leq& \frac{1}{2}.\end{aligned}$$

so $\sqrt{\frac{1.001\mathsf{deg}\ell}{|\mathbb{F}|}} < \sqrt{1.001/2} < 0.71 < 0.75 < \gamma$. Also see that

$$\begin{aligned}\frac{\ell\mathsf{deg}}{|\mathbb{F}|} \leq& \frac{15}{100\ell}\\ \ell^2 \leq& \frac{3|\mathbb{F}|}{20\mathsf{deg}}\\ \ell \leq& \sqrt{\frac{3|\mathbb{F}|}{20\mathsf{deg}}}.\end{aligned}$$

By Lemma 5.7 there exists $\mathcal{L}$ that is a below $\epsilon|\mathbb{F}|^{a\mathsf{dim}}/12 = \alpha n$ correcting lines set for $\mathbb{F}^{a\mathsf{dim}}$ with accuracy error $11\epsilon/12$, *strong* confidence error

$$O\left(\frac{a^{2a+1}400^a}{|\mathbb{F}|^a\epsilon^2}\right),$$

and line count $|\mathbb{F}|$. Further the size of the final correcting lines set is

$$|\mathcal{L}| = |\mathbb{F}|^{a(\mathsf{dim}+2a+O(1))}\mathsf{poly}(\mathsf{dim})$$

and $\mathcal{L}$ is time and space $\mathsf{poly}(|\mathbb{F}|^a\mathsf{dim})$ uniform.

By Lemma 5.8 there is a $q = |\mathbb{F}|(|\mathbb{F}|-1)+1 \leq |\mathbb{F}|^2$ query, time

$$|\mathbb{F}|\left(\mathsf{poly}(|\mathbb{F}|^a\mathsf{dim}) + O\left(|\mathbb{F}|\left(a\mathsf{dim}\mathsf{polylog}(|\mathbb{F}|) + \mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)\right)\right)\right) = \mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$$

space

$$\mathsf{poly}(|\mathbb{F}|^a\mathsf{dim}) + O((a\mathsf{dim}+|\mathbb{F}|)\log(|\mathbb{F}|)) + |\mathbb{F}|\mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right) = \mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$$

below αn, factor $\delta = O\left(\frac{a^{2a+1}400^a}{|\mathbb{F}|^a\epsilon^2}\right)$ list improving set, $\mathcal{I}$, for C with size $|\mathcal{I}| = \frac{|\mathcal{L}|}{|\mathbb{F}^{a\mathsf{dim}}|} = |\mathbb{F}|^{a(2a+O(1))}\mathsf{poly}(\mathsf{dim})$, out-of-codeword tolerance $\gamma - 11\epsilon/12 - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}$, and input list length ℓ. Further since $\gamma = 1-24\alpha \geq 1-\frac{1}{4\ell}$ and $\ell \leq \sqrt{\frac{3|\mathbb{F}|}{20\mathsf{deg}}}$, then the output list length is also ℓ.

Then by Lemma 5.4 there is an algorithm D that takes as input an ℓ-multi-string w and outputs the description of a function f such that if for some codeword multi-string Q we have that

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \alpha$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq \gamma - 11\epsilon/12 - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}$$

then the function f has $f(w) = Q$. See that $1-50\alpha \leq \gamma - 11\epsilon/12 - \frac{\ell\mathsf{deg}}{|\mathbb{F}|}$, so the second condition holds if

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1-50\alpha.$$

Let $k = \lceil \log(n+1)/\log(1/6\delta)\rceil$. Before we give the efficiency of D and f, we first simplify k and q^k. See that

$$\begin{aligned}
\log(1/6\delta) =& \log\left(O\left(\frac{|\mathbb{F}|^a\epsilon^2}{a^{2a+1}400^a}\right)\right)\\
=& 2\log(\epsilon) + a\log(|\mathbb{F}|) - (2a+1)(\log(a)+O(1)) + O(1)\\
=& a\log(|\mathbb{F}|) - O(a\log(a) + \log(1/\epsilon))\\
\geq& a\log(|\mathbb{F}|)/2
\end{aligned}$$

as long as $\log(|\mathbb{F}|)$ is sufficiently larger than $\log(a)+\log(1/\epsilon)$. Then we can simplify k to

$$k = O\left(\frac{\log(|\mathbb{F}^{a\mathsf{dim}}|)}{\log(1/6\delta)}\right) = O(\mathsf{dim}).$$

See that

$$\begin{aligned}
q^k \leq& qn^{\log(q)/\log(1/6\delta)}\\
\leq& |\mathbb{F}|^2 n^{\frac{2\log(|\mathbb{F}|)}{a\log(|\mathbb{F}|)/2}}\\
\leq& |\mathbb{F}|^2 n^{4/a}.
\end{aligned}$$

Then the algorithm f has description length $O(k \log(|\mathbb{F}|)) = O(\mathsf{dim}(\log(|\mathbb{F}|)))$, and each output location can be computed with $q^k \leq |\mathbb{F}|^2 n^{4/a}$ queries in time

$$O(Tq^k) = O(\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)|\mathbb{F}|^2 n^{4/a}) = n^{4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$$

and space

$$O(kS) = \mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell).$$

The algorithm D runs with

$$O(nq^k|\mathcal{I}|^2) = O(n^{1+4/a}|\mathbb{F}|^2|\mathbb{F}|^{2a(2a+O(1))}\mathsf{poly}(\mathsf{dim})) = n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$$

queries in time

$$O(nTq^k|\mathcal{I}|^2) = n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}\ell),$$

and space $O(kS + \log(n|\mathcal{I}||\mathbb{F}|)) = \mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$.

Finally, to complete the proof, we apply Lemma 5.9 to convert the output of f into ℓ individual functions $f_1, \ldots, f_\ell$ that output $Q_1, \ldots, Q_\ell$. This last step uses an additional nq^k queries and runs in time $n^{1+4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$ and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$. The final f_i needs to query the input $|\mathbb{F}|q^k = n^{4/a}\mathsf{poly}(|\mathbb{F}|)$ times and run in time $n^{4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$ and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$ to compute a single symbol of its output. □

5.5 Why Not Use Local Correctors from Low Error List Recovery for High Error?

One might suspect that one can use the same correctors for the low error list recovery setting to list decode in the high error setting, just use low agreement instead of high agreement in the local correction. By sampling properties, this will still include all the nearby codeword symbols at most locations even in the high error setting. So we corrected most codewords, there are fewer errors, so we are done, right?

Unfortunately, there are two kinds of errors. In codeword errors where we don't decode symbols when they are in a nearby codeword, and out-of-codeword errors, were we decode symbols that are not in a nearby codeword. Our sampling property just gives us fewer in codeword errors. Without some sort of high error (low agreement) low degree testing properties for our local correctors, we don't know how to rule out many of these out-of-codeword type errors.

What is so bad about out-of-codeword errors? Recall that in our iterative correction paradigm, we start with some number of errors, and a family of local correctors that decreases the number of errors on average. Then we find a specific local corrector that decreases the number of errors. To choose such a local corrector, we need to test every local corrector. These out-of-codeword errors make it harder to test how good a local corrector is. A local corrector that "corrects" a lot of symbols that are not in a codeword could look better than it really is.

One approach to testing the corrector is to take all local correctors and take a vote on which symbols are right. Only symbols from the overwhelming majority would be put in the good list. Then we take whichever corrector outputs the good list most often. Unfortunately, without any bound on the out-of-codeword errors, the out-of-codeword errors may collude to prevent us from reducing the number of in-codeword errors very far below $\sqrt{n}$ without using more time or space.

One potential way to overcome this is to show that if out-of-codeword errors do have this sort of collusion, then they must actually be from other codewords, and then we can handle them again through sampling. This is essentially what we do, but *only* at the first step. Our first local correctors comes from a high error low degree test. This forces us to have few out-of-codeword errors, since if there are many out-of-codeword errors that would imply they came from a nearby codeword and thus are not out-of-codeword. After that, there are few out-of-codeword errors, and sampling suffices for the rest of the analysis. In fact, we use such a local corrector in the next section.

6 High Error List Decoding to Low Error List Recovery

Now to reduce from the high error regime to the low error regime, we need a randomness efficient low degree test for Reed-Muller codes that works in the high error, or low agreement, regime. The structure of the test needs to guarantee we can do local list recovery if a string is near a codeword, and the soundness of the test needs to guarantee that we don't often decode extra symbols that aren't from a nearby codeword.

6.1 Low Degree Testing

The local tester we use is the space vs point test by Moshkovitz and Raz [MR06] which is based on sampling two directions from a small subfield, and then looking at the subspace through those points and the point we want to decode. For notation, we will first introduce an affine 3 dimensional subspace.

Definition 6.1 (Affine Subspace)**.** *For a field $\mathbb{F}$, dimension dim, and linearly independent $z, y_1, y_2 \in \mathbb{F}^{\mathsf{dim}}$, denote by $\mathsf{affine}(0, z, y_1, y_2)$ the affine, dimension 3 subspace that contains $0, z, y_1,$ and y_2.*

More concisely, we also define the span of z, y_1, y_2 as $\mathsf{span}\{z, y_1, y_2\} = \mathsf{affine}(0, z, y_1, y_2)$.

The space versus point test takes two functions on a three dimensional subspace, one from a global function, f, and one that is low degree, g, and returns whether they are equal at a specific point. Importantly, g does not need to come from a global function.

Definition 6.2 (Space Vs. Point)**.** *Fix dimension $\mathsf{dim} \geq 3$, field $\mathbb{F}$, degree deg, and function $f : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$. Let g be a set of degree deg functions on three dimensional subspaces. For notation, for every linearly independent $z, y_1, y_2 \in \mathbb{F}^{\mathsf{dim}}$, there is a function $g_{\mathsf{span}\{z,y_1,y_2\}} : \mathsf{span}\{z, y_1, y_2\} \to \mathbb{F}$. Then we define $\mathsf{SpacePoint}$ as*

$$\mathsf{SpacePoint}^{f,g}(z, y_1, y_2) = \begin{cases} \text{true} & z, y_1, y_2 \text{ are linearly dependent} \\ g_{\mathsf{span}\{z,y_1,y_2\}}(z) = f(z) & \text{otherwise} \end{cases}.$$

Similarly, if for some integer ℓ we have $f : \mathbb{F}^{\mathsf{dim}} \to \binom{\mathbb{F}}{\ell}$ then we define

$$\mathsf{SpacePoint}^{f,g}(z, y_1, y_2) = \begin{cases} \text{true} & z, y_1, y_2 \text{ are linearly dependent} \\ g_{\mathsf{span}\{z,y_1,y_2\}}(z) \in f(z) & \text{otherwise} \end{cases}.$$

Often it is convenient to think of g as the closest degree d polynomial to f on a subspace, but in the list decoding setting, it will also be convenient to think of g as any close enough local codeword. Then the local testing result says that if f agrees with any choice of g very often, then in fact g itself must mostly come from a small set of global low degree codewords. In particular, this is useful because it means that any local corrections that succeed often must be from a nearby codeword. The following is [MR06, Theorem 2] (with some observations from [MR06, Lemma 6.1]).

Lemma 6.3 (Subspace from Subfield vs. Point Soundness)**.** *Fix dimension $\mathsf{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$ and a degree deg. Let C be the Reed-Muller code over field $\mathbb{F}$, dimension dim and degree deg. Denote $\eta = 2^7 \mathsf{dim} \left(\left(\frac{1}{|\mathbb{H}|} \right)^{1/8} + \left(\frac{\mathsf{dim} \cdot \mathsf{deg}}{|\mathbb{F}|} \right)^{1/4} \right)$. For every function $w : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$ and every γ if for some g we have*

$$\Pr_{z \in \mathbb{F}^m, y_1, y_2 \in \mathbb{H}^m} [\mathsf{SpacePoint}^{w,g}(z, y_1, y_2)] = \gamma$$

then the following holds:

Decoding *There exists a degree at most deg polynomial $Q : \mathbb{F}^{\mathsf{dim}} \to \mathbb{F}$ such that*

$$\Pr_{x \in \mathbb{F}^{\mathsf{dim}}} [Q(x) = w(x)] \geq \gamma - 3\eta.$$

List decoding [2] *For every* $\alpha \geq 3\eta + 4\sqrt{\frac{\mathsf{deg}}{|\mathbb{F}|}}$ *for* $\alpha' = \alpha - 3\eta - 2\sqrt{\frac{\mathsf{deg}}{|\mathbb{F}|}}$ *we have*

$$\Pr_{z \in \mathbb{F}^m, y_1, y_2 \in \mathbb{H}^m}[\neg \mathsf{SpacePoint}^{w,g}(z, y_1, y_2) \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w) \text{ s.t. } Q|_{\mathsf{span}\{z,y_1,y_2\}} \equiv g_{\mathsf{span}\{z,y_1,y_2\}}] \geq 1 - \alpha$$

where $\mathsf{near}^C_{\alpha'}(w)$ *is the set of codewords that agree with* w *on an* α' *fraction of locations, as in Definition 4.8.*

It is important that the nearby codewords explain g, not the original string w. This allows us to conclude that we cannot have many local correctors that output symbols not from a nearby codeword. Too much agreement means that there must be a codeword nearby that they agree with.

By [MR06, Corollary 5.8], there is a partial converse to Lemma 6.3. Namely that these subspaces are a sampler.

Lemma 6.4 (Subfields Sample)**.** *Let* $\mathsf{dim}, \mathbb{F}, \mathbb{H}$, *and* deg *be as in Lemma 6.3. Take any set* $A \subseteq \mathbb{F}^{\mathsf{dim}}$*, let* $\mu = \frac{|A|}{|\mathbb{F}^{\mathsf{dim}}|}$*, and any constant* $\epsilon > 0$*. Then sample* s *as the random subspace that goes through* 0*, a uniformly random* $z \in \mathbb{F}^{\mathsf{dim}}$*, and* $y_1, y_2 \in \mathbb{H}^{\mathsf{dim}}$ *that are also uniformly random and linear independent. Then we have :*

$$\Pr_s\left[\left|\frac{|s \cap A|}{|s|} - \mu\right| \geq \epsilon\right] \leq \frac{1}{\epsilon^2}\left(\frac{\mu}{|\mathbb{H}|} + \frac{1}{|\mathbb{F}|}\right).$$

6.2 Local List Testing

The prior results give us results for local list decoding, which is sufficient for our application. But, there is a straightforward reduction to get local list recovery from local list decoding. We provide this more general result in case it is needed in the future. The reader may skip this section and assume that $\ell = 1$ in the next section.

Lemma 6.5 (List Recovery From List Decoding)**.** *Let* C *be a code with codewords of length* n *and alphabet* Σ*. Let* S *be some set of subsets of* $[n]$*. Suppose that the following holds.*

List Decoding: *For some* α *and* α' *we have that for all families of functions* $(g : s \to \Sigma)_{s \in S}$ *and strings* $w \in \Sigma^n$ *that*

$$\Pr_{s \in S, i \in s}[g_s(i) \neq w_i \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w) \text{ s.t. } Q|_s \equiv g_s] \geq 1 - \alpha$$

where $\mathsf{near}^C_{\alpha'}(w)$ *is the set of codewords that agree with* w *on an* α' *fraction of locations, as in Definition 4.8.*

Then we have the following.

List Recovery: *For any* ℓ*-multi-string* $w' \in \binom{\Sigma}{\ell}^n$*, and all families of functions* $(g : s \to \Sigma)_{s \in S}$ *we have that*

$$\Pr_{s \in S, i \in s}[g_s(i) \notin w'_i \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \geq 1 - \ell\alpha.$$

Proof. The proof is through a greedy algorithm. Choose any length n string w, contained in the ℓ-multi-string w'. For this string, see that $\mathsf{near}^C_{\alpha'}(w) \subseteq \mathsf{near}^C_{\alpha'}(w')$. Therefore,

$$\Pr_{s \in S, i \in s}[g_s(i) \neq w_i \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \geq 1 - \alpha.$$

That is

$$\begin{aligned}
&\Pr_{s \in S, i \in s}[g_s(i) \neq w_i \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \\
=&\Pr_{s \in S, i \in s}[g_s(i) \neq w_i] + \\
&\Pr_{s \in S, i \in s}[g_s(i) = w_i] \Pr_{s \in S, i \in s \text{ where } g_s(i) = w_i}[\exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \\
\geq& 1 - \alpha.
\end{aligned}$$

[2] This result can be found by looking into the proof of [MR06, Lemma 6.1] and observing that its f is $f(\gamma) = \gamma - 3\eta$ and setting d' to deg.

Now taking the expectation over a random string w contained in w' we have

$$\begin{aligned}
1-\alpha \leq & 1 - \mathop{\mathbb{E}}_{s\in S, i\in s}\left[\frac{1_{g_s(i)\in w'_i}}{|w'_i|}\right] + \mathop{\mathbb{E}}_{s\in S, i\in s}\left[\frac{1_{g_s(i)\in w'_i}}{|w'_i|}\right] \Pr_{s\in S, i\in s \text{ where } g_s(i)\in w'_i}[\exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \\
\leq & 1 - \frac{1}{\ell}\Pr_{s\in S, i\in s}[g_s(i)\in w'_i] + \frac{1}{\ell}\Pr_{s\in S, i\in s}[g_s(i)\in w'_i] \Pr_{s\in S, i\in s \text{ where } g_s(i)\in w'_i}[\exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \\
1-\ell\alpha \leq & 1 - \Pr_{s\in S, i\in s}[g_s(i)\in w'_i] + \Pr_{s\in S, i\in s}[g_s(i)\in w'_i] \Pr_{s\in S, i\in s \text{ where } g_s(i)\in w'_i}[\exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s] \\
\leq & \Pr_{s\in S, i\in s}[g_s(i)\notin w'_i \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_s \equiv g_s].
\end{aligned}$$

□

Applying this to Lemma 6.3 we get the following

Corollary 6.6 (Subspace vs. Point List Recovery)**.** *Fix dimension $\mathsf{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$ and a degree deg. Let C be the Reed-Muller code over field $\mathbb{F}$, dimension dim and degree deg. Denote $\eta = 2^7 \mathsf{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{\mathsf{dim}\cdot\mathsf{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$. Then the following holds.*

List Recovery: *For every $\alpha \geq 3\eta + 4\sqrt{\frac{\mathsf{deg}}{|\mathbb{F}|}}$ let $\alpha' = \alpha - 3\eta - 2\sqrt{\frac{\mathsf{deg}}{|\mathbb{F}|}}$. For any ℓ-multi-string $w' \in \binom{\Sigma}{\ell}^n$, and all families of functions g we have that*

$$\Pr_{z\in\mathbb{F}^m, y_1, y_2 \in \mathbb{H}^m}[\neg \mathsf{SpacePoint}^{w',g}(z, y_1, y_2) \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w') \text{ s.t. } Q|_{\mathsf{span}\{z,y_1,y_2\}} \equiv g_{\mathsf{span}\{z,y_1,y_2\}}] \geq 1 - \ell\alpha.$$

This test allows us to rule out too many out-of-codeword errors. Specifically, every out-of-codeword error comes from a function g restricted to some span that has some agreement with the code. If there is too much such agreement, this must be because some of the agreement comes from a nearby codeword.

6.3 Local Correctors

Now we can prove that these subspaces are good local correctors. The low completeness error comes from the sampling property of subspaces, and the low soundness error comes from the list decoding property of the low degree test. For notation, recall that $\mathsf{near}^C_\gamma(w)$ is the set of codewords that are in w for γ fraction of coordinates, as defined in Definition 4.8.

Lemma 6.7 (Pseudorandom Local Correctors)**.** *Fix dimension $\mathsf{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$, an integer ℓ and a degree deg. Denote $\eta = 2^7 \mathsf{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{\mathsf{dim}\cdot\mathsf{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$.*

Then for any $\epsilon > 0$, relative agreement γ, and $c > 1$ such that $\gamma > \sqrt{\mathsf{deg}\ell c/|\mathbb{F}|}$, for some $L \leq \frac{\ell}{\gamma}\frac{1}{1-1/c}$, there is a set of local correctors $\mathcal{I}$ that compute any index of the output using only $|\mathbb{F}^3|$ queries to the input and time $|\mathbb{F}|^3 \mathsf{poly}\left(\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$. There are $|\mathcal{I}| \leq |\mathbb{H}|^{2\mathsf{dim}}$ functions. Further, these functions $\mathcal{I}$ are such that for any ℓ-string w we have:

Completeness: *For any $y \in \mathsf{near}^C_{\gamma+\epsilon}(w)$ we have*

$$\Pr_{f\in\mathcal{I}, i\in\mathbb{F}^{\mathsf{dim}}}[y_i \not\subseteq f(w)_i] \leq \frac{1}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right).$$

Soundness: *For any α such that $(\gamma/\ell)(1-\alpha) > 4\eta$ we have*

$$\Pr_{f\in\mathcal{I}, x\in\mathbb{F}^{\mathsf{dim}}}[f(w)_i \not\subseteq \mathsf{near\text{-}ms}^C_{(\gamma/\ell)(1-\alpha)-5\eta}(w)_i] \leq 1-\alpha.$$

Proof. Each local corrector is associated with linearly independent $y_1, y_2 \in \mathbb{H}^{\mathsf{dim}}$. To correct the symbol at $x \in \mathbb{F}^{\mathsf{dim}}$, it takes the three dimensional subspace through $0, x, y_1, y_2$ and does list recovery with agreement γ on that subspace. If x is already in the subspace spanned by y_1 and y_2, we give up and don't output anything. This kind of failure is why their is an extra $\frac{1}{|\mathbb{F}|}$ in the completeness error.

Completeness: Take any codeword $y \in \mathsf{near}^C_{\gamma+\epsilon}(w)$ so that, by definition, y agrees with w on at least $\gamma + \epsilon$ fraction of places. Let A be the locations where y is contained in w. Then by the sampling property, the probability we sample A at fewer than γ fraction of places is at most $\frac{1}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{1}{|\mathbb{F}|}\right)$. So the probability we under-sample A or we give up is at most $\frac{1}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right)$. By the list decoding of Lemma 4.21, our local corrector will output the symbols for y whenever we sample γ fraction of points from A. Thus the probability that we don't output a symbol of y is at most

$$\frac{1}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right).$$

Soundness: Suppose that for $(1-\alpha)$ fraction of local correctors and positions the local corrector outputs a symbol not in $\mathsf{near\text{-}ms}^C_{(\gamma/\ell)-\beta-5\eta}(w)$ for $\beta = \frac{\gamma\alpha}{\ell}$.

This implies that for a $(1-\alpha)$ fraction of local correctors and positions that there is a local codeword not from $\mathsf{near\text{-}ms}^C_{(\gamma/\ell)-\beta-5\eta}(w)$ that is contained in w for γ fraction of points. Let g be the family of functions outputting such local codewords. Then using g in $\mathsf{SpacePoint}$ with w, we get a test that succeeds with probability $\gamma(1-\alpha)$.

We will use Corollary 6.6, but with its α set slightly less than γ/ℓ. Let $\gamma' = \gamma/\ell - \beta - 3\eta - 3\sqrt{\mathsf{deg}/|\mathbb{F}|}$. See that

$$\begin{aligned}\gamma' + 3\eta + 2\sqrt{\mathsf{deg}/|\mathbb{F}|} =& \gamma/\ell - \beta - \sqrt{\mathsf{deg}/|\mathbb{F}|}\\ \geq& 4\eta - \sqrt{\mathsf{deg}/|\mathbb{F}|}\\ \geq& 3\eta + 4\sqrt{\mathsf{deg}/|\mathbb{F}|}\end{aligned}$$

Since $\eta \geq 5\sqrt{\mathsf{deg}/|\mathbb{F}|}$. Then by Lemma 6.3 we have that

$$\begin{aligned}&\Pr_{z\in\mathbb{F}^m, y_1,y_2\in\mathbb{H}^m}[\neg\mathsf{SpacePoint}^{w,g}(z,y_1,y_2) \vee \exists Q \in \mathsf{near}^C_{\gamma'}(w) \text{ s.t. } Q|_{\mathsf{span}\{z,y_1,y_2\}} \equiv g_{\mathsf{span}\{z,y_1,y_2\}}]\\ \geq& 1 - \ell(\gamma' + 3\eta + 2\sqrt{\mathsf{deg}/|\mathbb{F}|})\\ =& 1 - \ell(\frac{\gamma}{\ell} - \beta - \sqrt{\mathsf{deg}/|\mathbb{F}|})\\ >& 1 - \gamma + \alpha\gamma.\end{aligned}$$

Since the probability that $\mathsf{SpacePoint}^{w,g}(z,y_1,y_2)$ fails is at most $1-\gamma(1-\alpha) = 1-\gamma+\beta$ there must be some codeword Q in $\mathsf{near\text{-}ms}^C_{\gamma'}(w)$ (and thus in $\mathsf{near\text{-}ms}^C_{(\gamma/\ell)(1-\alpha)-5\eta}(w)$) that has $Q|_{\mathsf{span}\{z,y_1,y_2\}} \equiv g_{\mathsf{span}\{z,y_1,y_2\}}$. But this is a contradiction since we chose g specifically to not agree with any such Q. So there cannot be $1-\alpha$ fraction of local correctors that output a symbol not in $\mathsf{near\text{-}ms}^C_{(\gamma/\ell)(1-\alpha)-5\eta}(w)$.

The number of queries comes from the size of a three dimensional subspace, and the time comes from Lemma 4.21. □

Then as long as we choose γ, α and ϵ appropriately so that $\mathsf{near\text{-}ms}^C_{\gamma+\epsilon}(w) = \mathsf{near\text{-}ms}^C_{\gamma(1-\alpha)-5\eta}(w)$ then we in particular have that for most coordinates and correctors we recover all symbols in nearby codewords, and a not too small fraction of coordinates and functions only recover symbols in nearby codewords. So some corrector will be good enough to use with our low error list recovery algorithm from Lemma 5.10.

Corollary 6.8 (There Exists a Good Local Corrector)**.** *Fix dimension $\mathsf{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$ and a degree deg. Denote $\eta = 2^7\mathsf{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{\mathsf{dim}\cdot\mathsf{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$. Take any $\epsilon, \alpha > 0$, and relative agreement γ such that $\gamma(1-\alpha) > 4\eta$. Then there is some $L \leq \frac{1.01}{\gamma}$, such that for any string w with*

$$\mathsf{near\text{-}ms}^C_{\gamma+\epsilon}(w) = \mathsf{near\text{-}ms}^C_{\gamma(1-\alpha)-5\eta}(w),$$

for $\mathcal{I}$ the family of functions from Lemma 6.7 there is a local corrector $f \in \mathcal{I}$ such that

Completeness:

$$\Pr_{f\in\mathcal{I},i\in\mathbb{F}^{\mathsf{dim}}}[\mathsf{near\text{-}ms}^{C}_{\gamma+\epsilon}(w)_i \not\subseteq f(w)_i] \leq \frac{2L}{\alpha\epsilon^2}\left(\frac{1}{|\mathbb{H}|}+\frac{2}{|\mathbb{F}|}\right).$$

Soundness:

$$\Pr_{f\in\mathcal{I},x\in\mathbb{F}^{\mathsf{dim}}}[f(w)_i \not\subseteq \mathsf{near\text{-}ms}^{C}_{\gamma(1-\alpha)-5\eta}(w)_i] \leq 1-\frac{\alpha}{2}.$$

Proof. If this was a single inequality, then we could use the bounds from Lemma 6.7 and use that some f must meet them in expectation. However, since we have two bounds, we need to show that the probability of failing both is less than one.

Since we only have a soundness expectation of $1-\alpha$, we could have $1-\alpha$ fraction of $f \in \mathcal{I}$ output symbols not in a nearby codeword at every index. But we know that α fraction of the time, a random $f \in \mathcal{I}$ and a random index must not output any extra symbols. So let τ be the fraction of functions $f \in \mathcal{I}$ that outputs an extra symbol for more than $1-\alpha/2$ fraction of coordinates. Then $\tau(1-\alpha/2) < 1-\alpha$. Thus the soundness condition fails with probability at most $\tau < 1-\alpha/2$.

Now we need the completeness condition to fail with probability at most $\alpha/2$. By a union bound and a Markov inequality, for all but $\alpha/2$ fraction of $f \in \mathcal{I}$ does the completeness bound hold. Thus with probability less than one do both fail. Thus for some $f \in \mathcal{I}$ both the soundness and the completeness holds. □

Our final protocol will try all functions $f \in \mathcal{I}$ and many choices of γ until it finds a local corrector that works.

6.4 Why not Use Local Correctors From List Decoding in List Recovery?

One might ask why not use the local correctors from this section in the recursion for low error list recovery. The issue is that subspaces don't sample well enough for the recursive procedure to be efficient. If we tried to use such a local corrector for the entire decoding, then our decoder would not be time efficient. This is because a k dimensional subspace can only decrease the distance by a factor of $\frac{1}{|\mathbb{F}|^k}$ at the cost of increasing the number of queries by a factor $|\mathbb{F}|^k$. The number of queries multiplies at every level of recursion, just like the fraction of errors. If we tried to apply this recursively, we would get a quadratic time decoder. We need curves, manifolds, or many subspaces together.

However, such a local corrector is fine to use once at the beginning of correction since we only pay this overhead once. After which we can use our more query efficient local corrector to remove any remaining errors.

7 Uniform List Correction For Reed-Muller Codes

Now we can finally construct our list corrector for Reed-Muller codes. First, in Theorem 7.1 we will state our result for some conditions that are convenient to prove the result. Then in Theorem 7.2 we will show for what choices of γ and τ we can satisfy these conditions. Finally, we use code concatenation to get small alphabet for the special case where γ and τ are constants.

Theorem 7.1 (Uniform, Explicit, High Error List Correction for Reed-Muller Codes)**.** *There is some large constant c_0 such that the following holds. Take any field $\mathbb{F}$ with $|\mathbb{F}| \geq 101$, subfield $H \subseteq \mathbb{F}$, any dimensions dim and a, and numbers $\alpha, \gamma, \epsilon > 0$. Let $n = |\mathbb{F}^{a\mathsf{dim}}|$. Let C be the Reed-Muller code for field $\mathbb{F}$, dimension $a\mathsf{dim}$ and degree deg. Denote $\eta = 2^7 a\mathsf{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8}+\left(\frac{a\cdot\mathsf{dim}\cdot\mathsf{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$.*

Let $L = \lfloor\frac{2}{\gamma}\rfloor$. Suppose each of the following holds:

1. $\log(|\mathbb{F}|) \geq c_0(\log(a)+\log(1/\alpha))$
2. $\gamma \geq 10(\alpha+\epsilon/\gamma+5\eta/\gamma)$

3. $\frac{L}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|}+\frac{2}{|\mathbb{F}|}\right) \leq \frac{\alpha^2}{500}$

4. $\alpha < \frac{1}{15000L}$

5. $\frac{L\mathsf{deg}}{|\mathbb{F}|} < 3000\alpha$

Then there is an algorithm D that takes as input a string w and outputs the description of at most L functions $f_1, \ldots, f_L$ such that for any codeword $Q \in C$ with

$$\Pr_{i\in[n]}[Q_i = w_i] \geq \gamma$$

then for some $i \in [L]$ we have that $f_i(w) = Q$.

Further, D runs with

$$n^{1+4/a}L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$$

queries in time

$$n^{1+4/a}L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}L),$$

and space

$$\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L).$$

Each f_i can compute a single index of its output with $n^{4/a}\mathsf{poly}(|\mathbb{F}|)$ queries in time $n^{4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L)$ and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L)$.

Proof. First, we need to choose a setting of γ' such that $\gamma' \leq \gamma$ and $\mathsf{near}^C_{\gamma'+\epsilon}(w) = \mathsf{near}^C_{\gamma'(1-\alpha)-5\eta}(w)$ so we can apply Corollary 6.8 to reduce the high error list decoding setting to the low error list recovery setting. After that, we apply Lemma 5.10 to recover the original codewords. Since there are not too many choices of γ' and local correctors we consider in Corollary 6.8, we will try them all and use whichever succeeds.

To find γ', we will try for every i from 0 to L with the value $\gamma'_i = \gamma - i(\epsilon + \gamma\alpha + 5\eta)$. Now we will show that for some γ'_i that $\mathsf{near}^C_{\gamma'_i+\epsilon}(w) = \mathsf{near}^C_{\gamma'_i(1-\alpha)-5\eta}(w)$. This is equivalent to showing that for some γ'_i, there is no codeword whose relative agreement with w is within $[\gamma'_i(1-\alpha)-5\eta, \gamma'_i+\epsilon)$. This is true if each interval is disjoint and there are only L total codewords that agree with w on $\gamma'_L(1-\alpha)-5\eta$ places.

To see that they are all distinct, see that for any $i \in [L]$

$$\begin{aligned}
\gamma'_i + \epsilon =& \gamma - i(\epsilon + \gamma\alpha + 5\eta) + \epsilon \\
\leq& \gamma - (i-1)(\epsilon + \gamma\alpha + 5\eta) - \gamma\alpha - 5\eta \\
\leq& \gamma'_{i-1} - \gamma\alpha - 5\eta \\
<& \gamma'_{i-1}(1-\alpha) - 5\eta.
\end{aligned}$$

To see that there are at most L, see that

$$\begin{aligned}
\gamma'_L(1-\alpha) - 5\eta \geq& \gamma - (L+1)(\gamma\alpha + \epsilon + 5\eta) \\
\geq& \gamma - \frac{3}{\gamma}(\gamma\alpha + \epsilon + 5\eta) \\
>& \gamma - \frac{\gamma}{3} \\
=& \frac{2\gamma}{3}.
\end{aligned}$$

Further since $\gamma > 50\eta \geq 1000\sqrt{\mathsf{deg}/|\mathbb{F}|}$ by Lemma 4.10 we have that there are at most $\ell \leq \frac{1.001}{2\gamma/3} \leq L$ codewords with agreement $\gamma'_L(1-\alpha)-5\eta$ of w. So all $L+1$ intervals are distinct and there are only L codewords, so some interval must not have a codeword at that distance. Since we will try all intervals, for analysis let $\frac{2\gamma}{3} \leq \gamma' \leq \gamma$ be such an interval.

See that that $\gamma'(1-\alpha) \geq \frac{2}{3}\gamma(1-\alpha) > 4\eta$ and $\mathsf{near}^C_{\gamma+\epsilon}(w) = \mathsf{near}^C_{\gamma(1-\alpha)-5\eta}(w)$. Let $\alpha' = 150\alpha$. By Corollary 6.8 there is a family of functions $\mathcal{I}$ and for some $f \in \mathcal{I}$ we have that

Completeness:

$$\Pr_{f\in\mathcal{I},i\in\mathbb{F}^{\mathsf{dim}}}[\mathsf{near\text{-}ms}^C_{\gamma+\epsilon}(w)_i \not\subseteq f(w)_i] \leq \frac{2L}{\alpha\epsilon^2}\left(\frac{1}{|\mathbb{H}|}+\frac{2}{|\mathbb{F}|}\right) \leq \frac{\alpha}{160} < \alpha'.$$

Soundness: Further we have

$$\mathop{\mathbb{E}}_{f\in\mathcal{I},x\in\mathbb{F}^{\mathsf{dim}}}[f(w)_i \not\subseteq \mathsf{near\text{-}ms}^C_{\gamma(1-\alpha)-5\eta}(w)_i] \leq 1-\frac{\alpha}{2} = 1-50\alpha'.$$

Further f only makes $|\mathbb{F}^3|$ queries to the input and runs in time $|\mathbb{F}|^3\mathsf{poly}\left(\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$. There are $|\mathcal{I}| \leq |\mathbb{H}|^{2a\mathsf{dim}}$ functions. Since we will try every single $f\in\mathcal{I}$, for analysis, assume we chose the right one.

See that each of the following holds:

1. $\log(|\mathbb{F}|) \geq c_0(\log(a)+\log(1/\alpha'))$
2. $\alpha' < \frac{1}{100\ell}$
3. $\frac{\ell\mathsf{deg}}{|\mathbb{F}|} < 15\alpha'$

Then by Lemma 5.10 there is an algorithm D that takes as input an ℓ-multi-string w and outputs the description of ℓ functions $f_1,\ldots,f_\ell$ such that if for some ℓ-codeword multi-string Q we have that

$$\Pr_{i\in[n]}[Q_i \not\subseteq w_i] \leq \alpha'$$

and

$$\Pr_{i\in[n]}[w_i \not\subseteq Q_i] \leq 1-50\alpha'$$

then if $Q=\bigcup'_{j\in[\ell]}Q'^j$ for codewords Q'^j, then for every $j\in[\ell]$ there is a $j'\in[\ell]$ such that $f_{j'}(w)=Q'^j$. In particular, if we give $f(w)$ as input to D, it will output functions such that when they are composed with f, they give the codewords in $\mathsf{near}^C_{\gamma(1-\alpha)-5\eta}(w)$ which contains $\mathsf{near}^C_\gamma(w)$. The only change is that the number of queries has an extra $|\mathbb{F}^3|$ term, the time has an extra $q|\mathbb{F}|^3\mathsf{poly}\left(\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$ term and the space has an extra $|\mathbb{F}|^3\mathsf{poly}\left(\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$ term.

Finally, there is an extra time $L|\mathbb{H}|^{2a\mathsf{dim}}$ overhead to the time and number of queries since we need to try this for every choice of γ'_i and local corrector in Corollary 6.8. This gives a final number of queries of

$$n^{1+4/a}L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$$

time

$$n^{1+4/a}L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}L),$$

and space

$$\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L).$$

And each f_i can compute a single index of its output with $n^{4/a}\mathsf{poly}(|\mathbb{F}|)$ queries, time $n^{4/a}\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L)$ and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L)$. □

Setting these parameters appropriately, we can get time and space efficiently list decodable codes for a wide variety of parameters.

Theorem 7.2 (Good Codes With Time-Space Efficient Uniform List Decoding)**.** *Choose any functions $\tau(n)$ and $\gamma(n)$ such that for all n we have $\frac{1}{\tau(n)}$ is an integer, $\gamma(n) \geq \frac{1000}{n^{\tau(n)^4/16}\tau(n)^{1.5}}$, and $\gamma(n) \geq \frac{5\cdot10^4}{\tau(n)n^{\tau(n)^3/c_0}}$ where c_0 is the large constant from Lemma 5.10. Then there is a uniform, infinite family of codes with rate $(\gamma\tau)^{O(1/\tau^2)}$ with an alphabet of size $O(\log(n))$ that can be list decoded from $1-\gamma$ fraction of errors in time $n^{1+O(\tau)}$ and space $n^{O(\tau)}$.*

Proof. We will use Theorem 7.1 with the following settings. For convenience I will leave the n implicit and denote $\tau(n)$ and $\gamma(n)$ as τ and γ respectively.

Let $a = \frac{1}{\tau}$ and $\mathsf{dim} = a^2$. For q that is a power of 2, our code, C, will be the Reed-Muller over the field $\mathbb{F}_q$ with dimension $\mathsf{dim}a$ and degree $\mathsf{deg} = \lfloor \frac{\gamma^8 \tau^{15} q}{10^{20}} \rfloor$. See that C has codeword length $n = q^{a\mathsf{dim}} = q^{1/\tau^3}$. See that the rate of C is at least $\left(\frac{\mathsf{deg}}{a\mathsf{dim}|\mathbb{F}|}\right)^{a\mathsf{dim}} = (\mathsf{poly}(\gamma\tau))^{1/\tau^3} = (\gamma\tau)^{O(1/\tau^3)}$

Let $\mathbb{H}$ be a subfield of $\mathbb{F}$ such that $|H| = 2^{\lceil \log(q)/\tau \rceil} = q^{O(\tau)}$, the constant $\alpha = \frac{\gamma}{5 \cdot 10^4}$ and $\epsilon = \frac{2 \cdot 10^6}{\gamma \sqrt{|\mathbb{H}|\gamma}}$.

Now to apply Theorem 7.1, we need to satisfy various inequalities. We start by bounding the η from Lemma 6.3. This seems to be the weak link in our algorithm. If one wants to improve our results, we suggest to start by improving a local testing result with a smaller η.

$$\begin{aligned}
\eta =& 2^7 a\mathsf{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{a \cdot \mathsf{dim} \cdot \mathsf{deg}}{|\mathbb{F}|}\right)^{1/4}\right) \\
\leq& \frac{128}{\tau^3}\left(q^{-\tau/8} + \left(\frac{1}{\tau^3|\mathbb{F}|}\frac{\gamma^8\tau^{15}q}{10^{20}}\right)^{1/4}\right) \\
=& \frac{128}{\tau^3}\left(q^{-\tau/8} + \frac{\tau^3\gamma^2}{10^5}\right).
\end{aligned}$$

Now by assumption we have

$$\begin{aligned}
\gamma \geq& \frac{1000}{n^{\tau^4/16}\tau^{1.5}} \\
\gamma \geq& \frac{1000}{q^{\tau/16}\tau^{1.5}} \\
\frac{\gamma\tau^{1.5}}{10^3} \geq& q^{-\tau/16} \\
\frac{\gamma^2\tau^3}{10^6} \geq& q^{-\tau/8}.
\end{aligned}$$

Thus we conclude that

$$\begin{aligned}
\eta \leq& \frac{128}{\tau}\left(\frac{\gamma^2\tau}{10^6} + \frac{\tau\gamma^2}{10^5}\right) \\
\leq& \frac{15}{10^4}\gamma^2.
\end{aligned}$$

Now let us check all the conditions in Theorem 7.1.

1. Now we show $\log(|\mathbb{F}|) \geq c_0(\log(a) + \log(1/\alpha))$.

   See that $|\mathbb{F}| = q = n^{\tau^3}$, so $\log(|\mathbb{F}|) = \tau^3 \log(n)$. See that $\gamma \geq \frac{5 \cdot 10^4}{\tau n^{\tau^3/c_0}}$, so $\alpha = \frac{\gamma}{5 \cdot 10^4} \geq \frac{1}{\tau n^{\tau^3/c_0}}$, so $\log(1/\alpha) \leq \tau^3 \log(n)/c_0 - \log(1/\tau)$. Thus

   $$\begin{aligned}
   \log(|\mathbb{F}|) =& \tau^3 \log(n) \\
   =& \tau^3 \log(n) c_0/c_0 - c_0 \log(1/\tau) + c_0 \log(1/\tau) \\
   \geq& c_0(\log(1/\alpha) + \log(a)).
   \end{aligned}$$

2. Now we show $\gamma \geq 10(\alpha + \epsilon/\gamma + 5\eta/\gamma)$. We do this in three parts.

   (a) We show $\alpha \leq \frac{\gamma}{100}$. This is true since we defined α to be a very small multiple of γ.

(b) We show $\frac{\epsilon}{\gamma} \leq \frac{\gamma}{100}$. See that

$$\begin{aligned}
\epsilon =& \frac{2 \cdot 10^6}{\gamma \sqrt{|\mathbb{H}|\gamma}} \\
\frac{\epsilon}{\gamma} =& \frac{2 \cdot 10^6}{\gamma^{2.5} \sqrt{|\mathbb{H}|}} \\
\leq& \frac{2 \cdot 10^6}{\gamma^{2.5}} q^{-\tau/2} \\
\leq& \frac{2 \cdot 10^6}{\gamma^{2.5}} \left(\frac{\gamma^2 \tau}{10^6} \right)^4 \\
\leq& \frac{\gamma}{100}.
\end{aligned}$$

(c) Finally we show that $\frac{\eta}{\gamma} \leq \frac{\gamma}{100}$. To see this, see that we have already bounded η by $\eta \leq \frac{15}{10^4}\gamma^2$. Thus $\frac{\eta}{\gamma} \leq \frac{\gamma}{100}$.

Altogether these imply that $\gamma \geq 10(\alpha + \epsilon/\gamma + 5\eta/\gamma)$.

3. Now we need to show that $\frac{L}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right) \leq \frac{\alpha^2}{500}$. See that

$$\begin{aligned}
\frac{L}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right) \leq& \frac{2}{\gamma}\left(\frac{\gamma\sqrt{|\mathbb{H}|\gamma}}{2 \cdot 10^6}\right)^2 \frac{3}{|\mathbb{H}|} \\
\leq& \frac{3\gamma^2}{2 \cdot 10^{12}} \\
\leq& \frac{\alpha^2}{500}.
\end{aligned}$$

4. Now we show $\alpha < \frac{1}{15000L}$. This is true since $\frac{\gamma}{2} \leq \frac{1}{L}$ and since we chose α to be a very small multiple of γ.

5. Finally we show that $\frac{L\mathsf{deg}}{|\mathbb{F}|} < 3000\alpha$. See that

$$\begin{aligned}
\frac{L\mathsf{deg}}{|\mathbb{F}|} \leq& \frac{2}{\gamma}\frac{\gamma^8\tau^7 q}{10^{20}}\frac{1}{q} \\
\leq& \frac{\gamma}{10^{19}} \\
\leq& 3000\alpha.
\end{aligned}$$

Now by Theorem 7.1 we know that there is a decoder for C that runs in time

$$\begin{aligned}
n^{1+4/a} L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}L) =& n^{1+O(\tau)} q^{O(\tau/\tau^3)} q^{O(1/\tau^2)} \mathsf{poly}(1/\tau\gamma), \\
=& n^{1+O(\tau)} n^{O(\tau)} n^{O(\tau)} n^{O(\tau^4)} \\
=& n^{t+O(\tau)},
\end{aligned}$$

and space

$$\begin{aligned}
\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L) =& q^{O(1/\tau)}\mathsf{dim} \\
=& q^{O(1/\tau)}(1/\tau)^{O(1)} \\
=& n^{O(\tau)}.
\end{aligned}$$

□

Now as a corollary when τ and γ are constants we can get an asymptotically good code (with a size $O(\log(n))$ alphabet).

Theorem 7.3 (Good Codes With Time-Space Efficient Uniform List Decoding)**.** *For any constants $\tau > 0$ and $\gamma > 0$, there exists an infinite family of uniform asymptotically good codes with rate $(\gamma\tau)^{O(1/\tau^2)}$ and alphabet size $O(\log(n))$ that can be list decoded from $1-\gamma$ fraction of errors in time $n^{1+\tau}$ and space n^{τ}.*

Now we can use our list decoding algorithm to get list recovery, and then we can concatenate our code to get list decoding with a smaller alphabet. Using Lemma 4.12, we can get efficient list recovery for Reed-Muller codes.

Theorem 7.4 (Good Codes With Time-Space Efficient Uniform List Recovery)**.** *For any constants $\tau > 0$, $\ell > 0$, and $\gamma > 0$, there exists an infinite family of uniform asymptotically good codes with rate $(\gamma\tau/\ell)^{O(1/\tau^2)}$ and alphabet size $O(\log(n))$ that is list recoverable from γ fraction of agreement with length ℓ input lists in time $O(n^{1+\tau})$ and space $O(n^{\tau})$.*

Proof. First we use Theorem 7.3 to get an asymptotically good code C with alphabet size $O(\log(n))$ that can be list decoded from $1-\frac{\gamma}{\ell}$ fraction of errors in time $O(n^{1+\tau/2})$ and space $O(n^{\tau/2})$. Now applying Lemma 4.12 we have that a list recovery algorithm for C with input length ℓ that recovers from γ fraction of agreement in time $O(\ell n^{1+\tau/2}\log(\log(n))) = O(n^{1+\tau})$ and space $O(\log(\ell\log(n)) + n^{\tau/2}) = O(n^{1+\tau})$. □

Now that we have list recovery of Reed-Muller codes, we can use code concatenation to get a code with a small alphabet that is list decodable. By concatenating Theorem 7.4 with Lemma 4.15 we get the following.

Theorem 7.5 (Constant Alphabet Codes With Time-Space Efficient List Decoding)**.** *For any constants $\epsilon > 0$, and $\tau > 0$, there exists an infinite family of uniform asymptotically good codes with rate $(\epsilon\tau)^{O(1/\tau^2)}$ and a size $(1/\epsilon)^{O(1/\epsilon)}$ alphabet which can be list decoded from $1-\epsilon$ fraction of errors in time $O(n^{1+\tau})$ and space $O(n^{\tau})$.*

Proof. By Lemma 4.15, there is a code C_2 with constant rate and alphabet $(1/\epsilon)^{O(1/\epsilon)}$ that can be encoded in time $\mathsf{poly}(n/\epsilon)$ and list decoded from $(1-\epsilon/2)$-fraction of errors with list size at most $\ell' = \exp(\mathsf{poly}(1/\epsilon))$ in time $\mathsf{poly}(\ell' n)$. By Theorem 7.4, there is an asymptotically good code C_1 with rate $(\epsilon\tau)^{O(1/\tau^2)}$ and alphabet size $O(\log(n))$ that can be list recovered from $1-\epsilon/2$ fraction of errors with input list length ℓ' in time $n^{1+\tau/2}$ and space $n^{\tau/2}$. Concatenating these two codes where C_2 has a length $O(\log(n))$ input, we get an asymptotically good code with rate $(\epsilon\tau)^{O(1/\tau^2)}$ and an alphabet of size $(1/\epsilon)^{O(1/\epsilon)}$ that can be list decoded from $(1-\epsilon/2)^2 \geq 1-\epsilon$ fraction of errors in time $n^{1+\tau/2}\mathsf{poly}(\log(n)\exp(\mathsf{poly}(1/\epsilon))) = n^{1+\tau}$ and space $n^{\tau/2} + \mathsf{poly}(\exp(\mathsf{poly}(1/\epsilon))\log(n)) = O(n^{\tau})$. □

In a similar way, we can use concatenation with the binary code form Lemma 4.16 to get a binary code that is list decodable with error close to one half.

Theorem 7.6 (Binary Codes With Time-Space Efficient List Decoding)**.** *For any constants $\epsilon > 0$, and $\tau > 0$, there exists an infinite family of uniform asymptotically good codes with rate $(\epsilon\tau)^{O(1/\tau^2)}$ and a binary alphabet which can be list decoded from $\frac{1}{2}-\epsilon$ fraction of errors in time $O(n^{1+\tau})$ and space $O(n^{\tau})$.*

Proof. By Lemma 4.16, there is a code C_2 with rate $\Omega(\epsilon^4)$ and a binary alphabet that can be list decoded from $\frac{1}{2}-\epsilon/2$ fraction of errors in time $\mathsf{poly}(n/\epsilon)$. They can also be encoded in time $\mathsf{poly}\left(n2^{\mathsf{poly}(1/\epsilon)}\right)$. By Theorem 7.4, there is an asymptotically good code C_1 with rate $(\epsilon\tau)^{O(1/\tau^2)}$ and alphabet size $O(\log(n))$ that can be list recovered from $1-\epsilon/2$ fraction of errors with input list length ℓ' in time $n^{1+\tau/2}$ and space $n^{\tau/2}$. Concatenating these two codes where C_2 has a length $O(\log(n))$ input, we get an asymptotically good code with rate $\Omega\left(\epsilon^4(\epsilon\tau)^{O(1/\tau^2)}\right) = (\epsilon\tau)^{O(1/\tau^2)}$ and a binary that can be list decoded from $(1/2-\epsilon/2)(1-\epsilon/2) \geq \frac{1}{2}-\epsilon$ fraction of errors in time $n^{1+\tau/2}\mathsf{poly}(\log(n)2^{\mathsf{poly}(1/\epsilon)}) = O(n^{1+\tau})$ and space $n^{\tau/2}+\mathsf{poly}(\log(n)2^{\mathsf{poly}(1/\epsilon)}) = O(n^{\tau})$. □

8 Open Problems

This is the first work showing that the time and space efficient deterministic list decoding is possible. Many problems remain in this area. Such as:

1. Find codes that are both time and space efficient to encode and time and space efficient to decode. A prior result [CM24] showed that there are good codes that are time and space efficient to encode. This result shows that even list decoding can be done time and space efficiently. However, these are different codes.

2. Get almost linear time and sub-polynomial space decoders for an asymptotically good code. Our code gets worse rate as our time and space improve. The prior work [CM25] achieved this for unique decoding for lifted Reed-Solomon codes. Could one get a similar result for the list decoding setting?

3. Make this code practical. We did not optimize our algorithm, and as a result we have astronomical constants like 10^{20} in our result. While some improvement can likely be made with some straightforward optimization, we suspect that to make this algorithm practical will require improved derandomized low degree tests. The local test of [MR06] alone requires a field size of over 10^{16} to get any nontrivial results.

   A related problem is to improve our rate, decoding radius trade off. In our results, to go from an agreement γ decoder to an agreement $\gamma/2$ decoder we have to decrease the rate of the code (in Theorem 7.2) by a factor $2^{-8\mathsf{dim}}$ where dim is the dimension of the Reed-Muller code. The Johnson bound suggests that we should be able to achieve this by decreasing the rate by a factor of $2^{-2\mathsf{dim}}$.

   Improving either of these results seems to require improved, randomness efficient low degree testing. While there has been much work on low degree testing, little of it has been in the randomness efficient setting. To the best of our knowledge [MR06] provides the most efficient known low degree test that is *both* in the low randomness and the low agreement regime.

4. Give non-uniform time and space efficient decoders for all local list correctable codes. Our results only give us time and space efficient decoders for codes that are both local list correctors *and* locally testable. This local testing property was not required in the unique decoding case since all locally correctable codes have a weaker testing property called vicinity local testing. If one is not trying to decode all the way from the decoding radius, this test suffices. But in the list correcting setting where one is trying to decode to almost the full distance of the code, vicinity local testing is not enough.

   It is not obvious whether all locally list correctable codes *should* be time and space efficiently decodable. The local testing property seems extremely important for our proof technique. So we more generally ask whether all locally list correctable codes are time and space efficiently list decodable (up to near the distance of the code). If any were not, then this would be a problem solvable by super linear time randomized algorithms provably in less time space product complexity than possible for a deterministic algorithm.

5. Construct simultaneously time and space efficient PRGs. List decoding was used in prior PRGs [STV01; SU05], and recent work has analyzed both the fine grained space complexity of PRGs [DT23; DPT24] and the fine grained time complexity of PRGs [CT21; CT22; Dor+22]. It is natural to attempt to get derandomization that is both very space efficient and very time efficient simultaneously.

   Since list decoding is a common ingredient in PRGs, we suspect that time and space efficient list decodable codes could be useful. If list decoding is used in such constructions, we suspect that to get optimal time and space efficiency we would need codes that are both time and space efficient to encode as well as to decode.

References

[AS97] Sanjeev Arora and Madhu Sudan. "Improved Low-Degree Testing and Its Applications". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, 485–495. ISBN: 0897918886. DOI: 10.1145/258533.258642. URL: https://doi.org/10.1145/258533.258642.

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic Checking of Proofs: A New Characterization of NP". In: *J. ACM* 45.1 (Jan. 1998), 70–122. ISSN: 0004-5411. DOI: 10.1145/273865.273901. URL: https://doi.org/10.1145/273865.273901.

[Ale05] M. Alekhnovich. "Linear diophantine equations over polynomials and soft decoding of Reed-Solomon codes". In: *IEEE Transactions on Information Theory* 51.7 (2005), pp. 2257–2265. DOI: 10.1109/TIT.2005.850097.

[Aro+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (May 1998), 501–555. ISSN: 0004-5411. DOI: 10.1145/278298.278306. URL: https://doi.org/10.1145/278298.278306.

[BDLN17] Amey Bhangale, Irit Dinur, and Inbal Livni Navon. "Cube vs. Cube Low Degree Test". In: *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Ed. by Christos H. Papadimitriou. Vol. 67. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 40:1–40:31. ISBN: 978-3-95977-029-3. DOI: 10.4230/LIPIcs.ITCS.2017.40. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2017.40.

[BM05] L.M.J. Bazzi and S.K. Mitter. "Endcoding complexity versus minimum distance". In: *IEEE Transactions on Information Theory* 51.6 (2005), pp. 2103–2112. DOI: 10.1109/TIT.2005.847727.

[BMS09] Louay Bazzi, Mohammad Mahdian, and Daniel A. Spielman. "The Minimum Distance of Turbo-Like Codes". In: *IEEE Transactions on Information Theory* 55.1 (2009), pp. 6–15. DOI: 10.1109/TIT.2008.2008114.

[BS06] Eli Ben-Sasson and Madhu Sudan. "Robust locally testable codes and products of codes". In: *Random Struct. Algorithms* 28.4 (July 2006), 387–402. ISSN: 1042-9832.

[Bab+91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking Computations in Polylogarithmic Time". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*. STOC '91. New Orleans, Louisiana, USA: Association for Computing Machinery, 1991, 21–32. ISBN: 0897913973. DOI: 10.1145/103418.103428. URL: https://doi.org/10.1145/103418.103428.

[Ben+03] Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. "Randomness-efficient low degree tests and short PCPs via epsilon-biased sets". In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: Association for Computing Machinery, 2003, 612–621. ISBN: 1581136749. DOI: 10.1145/780542.780631. URL: https://doi.org/10.1145/780542.780631.

[CM24] Joshua Cook and Dana Moshkovitz. "Explicit Time and Space Efficient Encoders Exist Only with Random Access". In: *The Proceedings of the 39th Computational Complexity Conference (CCC24)* (2024). URL: https://eccc.weizmann.ac.il/report/2024/032/.

[CM25] Joshua Cook and Dana Moshkovitz. "Time and Space Efficient Deterministic Decoders". In: *STOC'25* (2025). URL: https://eccc.weizmann.ac.il/report/2024/110/.

[CT21] Lijie Chen and Roei Tell. "Simple and fast derandomization from very hard functions: eliminating randomness at almost no cost". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. Virtual, Italy: Association for Computing Machinery, 2021, 283–291. ISBN: 9781450380539. DOI: 10.1145/3406325.3451059. URL: https://doi.org/10.1145/3406325.3451059.

[CT22] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 125–136. DOI: 10.1109/FOCS52979.2021.00021.

[Cou01] Nicolas T. Courtois. "Efficient Zero-Knowledge Authentication Based on a Linear Algebra Problem MinRank". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 402–421. ISBN: 978-3-540-45682-7.

[DG08] Irit Dinur and Elazar Goldenberg. "Locally Testing Direct Product in the Low Error Range". In: *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. 2008, pp. 613–622. DOI: 10.1109/FOCS.2008.26.

[DH09] Irit Dinur and Prahladh Harsha. "Composition of Low-Error 2-Query PCPs Using Decodable PCPs". In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. 2009, pp. 472–481. DOI: 10.1109/FOCS.2009.8.

[DPT24] Dean Doron, Edward Pyne, and Roei Tell. "Opening Up the Distinguisher: A Hardness to Randomness Approach for BPL=L That Uses Properties of BPL". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. STOC 2024. Vancouver, BC, Canada: Association for Computing Machinery, 2024, 2039–2049. ISBN: 9798400703836. DOI: 10.1145/3618260.3649772. URL: https://doi.org/10.1145/3618260.3649772.

[DT23] Dean Doron and Roei Tell. "Derandomization with Minimal Memory Footprint". In: *38th Computational Complexity Conference (CCC 2023)*. Ed. by Amnon Ta-Shma. Vol. 264. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 11:1–11:15. ISBN: 978-3-95977-282-2. DOI: 10.4230/LIPIcs.CCC.2023.11. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CCC.2023.11.

[Din+22] Irit Dinur, Shai Evra, Ron Livne, Alexander Lubotzky, and Shahar Mozes. "Locally testable codes with constant rate, distance, and locality". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, 357–374. ISBN: 9781450392648. DOI: 10.1145/3519935.3520024. URL: https://doi.org/10.1145/3519935.3520024.

[Dor+22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. "Nearly Optimal Pseudorandomness from Hardness". In: *J. ACM* 69.6 (Nov. 2022). ISSN: 0004-5411. DOI: 10.1145/3555307. URL: https://doi.org/10.1145/3555307.

[Eli57] Peter Elias. "List decoding for noisy channels". In: *Technical report (Massachusetts Institute of Technology. Research Laboratory of Electronics), No 335* (1957).

[FB98] Weishi Feng and R.E. Blahut. "Some results on the Sudan algorithm [for decoding Reed-Solomon codes]". In: *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No.98CH36252)*. 1998, pp. 57–. DOI: 10.1109/ISIT.1998.708638.

[FS95] Katalin Friedl and Madhu Sudan. "Some improvements to total degree tests". In: *Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'95)*. ISTCS '95. USA: IEEE Computer Society, 1995, p. 190. ISBN: 0818669152.

[Fei+91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating Clique is Almost NP-Complete (Preliminary Version)". In: *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. IEEE Computer Society, 1991, pp. 2–12. DOI: 10.1109/SFCS.1991.185341. URL: https://doi.org/10.1109/SFCS.1991.185341.

[GI03] Venkatesan Guruswami and Piotr Indyk. "Linear time encodable and list decodable codes". In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: Association for Computing Machinery, 2003, 126–135. ISBN: 1581136749. DOI: 10.1145/780542.780562. URL: https://doi.org/10.1145/780542.780562.

[GK16] Alan Guo and Swastik Kopparty. "List-Decoding Algorithms for Lifted Codes". In: *IEEE Transactions on Information Theory* 62.5 (2016), pp. 2719–2725. DOI: 10.1109/TIT.2016.2538766.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). ISSN: 0004-5411. DOI: 10.1145/2699436. URL: https://doi.org/10.1145/2699436.

[GKS13] Alan Guo, Swastik Kopparty, and Madhu Sudan. "New affine-invariant codes from lifting". In: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS '13. Berkeley, California, USA: Association for Computing Machinery, 2013, 529–540. ISBN: 9781450318594. DOI: 10.1145/2422436.2422494. URL: https://doi.org/10.1145/2422436.2422494.

[GM16] Ofer Grossman and Dana Moshkovitz. "Amplification and Derandomization without Slowdown". In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 770–779. DOI: 10.1109/FOCS.2016.87.

[GR22] Zeyu Guo and Noga Ron-Zewi. "Efficient List-Decoding With Constant Alphabet and List Sizes". In: *IEEE Transactions on Information Theory* 68.3 (2022), pp. 1663–1682. DOI: 10.1109/TIT.2021.3131992.

[GRS23] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. *Essential Coding Theory*. 2023. URL: https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/.

[GS06] Oded Goldreich and Madhu Sudan. "Locally testable codes and PCPs of almost-linear length". In: *J. ACM* 53.4 (July 2006), 558–655. ISSN: 0004-5411. DOI: 10.1145/1162349.1162351. URL: https://doi.org/10.1145/1162349.1162351.

[GS99] V. Guruswami and M. Sudan. "Improved decoding of Reed-Solomon and algebraic-geometry codes". In: *IEEE Transactions on Information Theory* 45.6 (1999), pp. 1757–1767. DOI: 10.1109/18.782097.

[Gol10] Oded Goldreich. *A Primer on Pseudorandom Generators*. University lecture series. American Mathematical Soc., 2010. ISBN: 978-0-8218-5192-0. URL: https://www.wisdom.weizmann.ac.il/~oded/prg-primer.html.

[Gop+18] Sivakanth Gopi, Swastik Kopparty, Rafael Oliveira, Noga Ron-Zewi, and Shubhangi Saraf. "Locally Testable and Locally Correctable Codes approaching the Gilbert-Varshamov Bound". In: *IEEE Transactions on Information Theory* 64.8 (2018), pp. 5813–5831. DOI: 10.1109/TIT.2018.2809788.

[Gro06] André Gronemeier. "A Note on the Decoding Complexity of Error-Correcting Codes". In: *Inf. Process. Lett.* 100.3 (2006), 116–119. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2006.06.006. URL: https://doi.org/10.1016/j.ipl.2006.06.006.

[Gur+02] V. Guruswami, J. Hastad, M. Sudan, and D. Zuckerman. "Combinatorial bounds for list decoding". In: *IEEE Transactions on Information Theory* 48.5 (2002), pp. 1021–1034. DOI: 10.1109/18.995539.

[Gur07] Venkatesan Guruswami. "Algorithmic results in list decoding". In: *Found. Trends Theor. Comput. Sci.* 2.2 (Jan. 2007), 107–195. ISSN: 1551-305X. DOI: 10.1561/0400000007. URL: https://doi.org/10.1561/0400000007.

[Gur09] Venkatesan Guruswami. "List Decoding of Binary Codes—A Brief Survey of Some Recent Results". In: *Proceedings of the 2nd International Workshop on Coding and Cryptology*. IWCC '09. Zhangjiajie, China: Springer-Verlag, 2009, 97–106. ISBN: 9783642018138. DOI: 10.1007/978-3-642-01877-0_10. URL: https://doi.org/10.1007/978-3-642-01877-0_10.

[HOW13] Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. "Local Correctability of Expander Codes". In: *Automata, Languages, and Programming*. Ed. by Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 540–551. ISBN: 978-3-642-39206-1.

[HRZW17] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. "Local List Recovery of High-Rate Tensor Codes & Applications". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 2017, pp. 204–215. DOI: 10.1109/FOCS.2017.27.

[Ham50] R. W. Hamming. "Error detecting and error correcting codes". In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[Har+24] Prahladh Harsha, Mrinal Kumar, Ramprasad Saptharishi, and Madhu Sudan. "An Improved Line-Point Low-Degree Test*". In: *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*. 2024, pp. 1883–1892. DOI: 10.1109/FOCS61266.2024.00113.

[HÅ01] Johan Håstad. "Some optimal inapproximability results". In: *J. ACM* 48.4 (July 2001), 798–859. ISSN: 0004-5411. DOI: 10.1145/502090.502098. URL: https://doi.org/10.1145/502090.502098.

[IKW09] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. "New direct-product testers and 2-query PCPs". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, 131–140. ISBN: 9781605585062. DOI: 10.1145/1536414.1536435. URL: https://doi.org/10.1145/1536414.1536435.

[Joh62] S. Johnson. "A new upper bound for error-correcting codes". In: *IRE Transactions on Information Theory* 8.3 (1962), pp. 203–207. DOI: 10.1109/TIT.1962.1057714.

[KSY14] Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. "High-rate codes with sublinear-time decoding". In: *J. ACM* 61.5 (2014). ISSN: 0004-5411. DOI: 10.1145/2629416. URL: https://doi.org/10.1145/2629416.

[KT00] Jonathan Katz and Luca Trevisan. "On the efficiency of local decoding procedures for error-correcting codes". In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*. STOC '00. Portland, Oregon, USA: Association for Computing Machinery, 2000, 80–86. ISBN: 1581131844. DOI: 10.1145/335305.335315. URL: https://doi.org/10.1145/335305.335315.

[KTS22] Dan Karliner and Amnon Ta-Shma. "Improved Local Testing for Multiplicity Codes". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Ed. by Amit Chakrabarti and Chaitanya Swamy. Vol. 245. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 11:1–11:19. ISBN: 978-3-95977-249-5. DOI: 10.4230/LIPIcs.APPROX/RANDOM.2022.11. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.APPROX/RANDOM.2022.11.

[KU06] Shankar Kalyanaraman and Christopher Umans. "On obtaining pseudorandomness from error-correcting codes". In: FSTTCS'06. Kolkata, India: Springer-Verlag, 2006, 105–116. ISBN: 3540499946. DOI: 10.1007/11944836_12. URL: https://doi.org/10.1007/11944836_12.

[KV03] R. Koetter and A. Vardy. "Algebraic soft-decision decoding of Reed-Solomon codes". In: *IEEE Transactions on Information Theory* 49.11 (2003), pp. 2809–2825. DOI: 10.1109/TIT.2003.819332.

[Kop+17] Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. "High-Rate Locally Correctable and Locally Testable Codes with Sub-Polynomial Query Complexity". In: *J. ACM* 64.2 (2017). ISSN: 0004-5411. DOI: 10.1145/3051093. URL: https://doi.org/10.1145/3051093.

[Kop+23] Swastik Kopparty, Noga Ron-Zewi, Shubhangi Saraf, and Mary Wootters. "Improved List Decoding of Folded Reed-Solomon and Multiplicity Codes". In: *SIAM Journal on Computing* 52.3 (2023), pp. 794–840. DOI: 10.1137/20M1370215. eprint: https://doi.org/10.1137/20M1370215. URL: https://doi.org/10.1137/20M1370215.

[LH22] Ting-Chun Lin and Min-Hsiu Hsieh. "c3-Locally Testable Codes from Lossless Expanders". In: *2022 IEEE International Symposium on Information Theory (ISIT)*. 2022, pp. 1175–1180. DOI: 10.1109/ISIT50566.2022.9834679.

[Lun+90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. "Algebraic methods for interactive proof systems". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 2–10 vol.1. DOI: 10.1109/FSCS.1990.89518.

[MR06] Dana Moshkovitz and Ran Raz. "Sub-constant error low degree test of almost-linear size". In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '06. Seattle, WA, USA: Association for Computing Machinery, 2006, 21–30. ISBN: 1595931341. DOI: 10.1145/1132516.1132520. URL: https://doi.org/10.1145/1132516.1132520.

[MR08] Dana Moshkovitz and Ran Raz. "Two-query PCP with subconstant error". In: *J. ACM* 57.5 (June 2008). ISSN: 0004-5411. DOI: 10.1145/1754399.1754402. URL: https://doi.org/10.1145/1754399.1754402.

[MZ23] Dor Minzer and Kai Zheng. "Approaching the Soundness Barrier: A Near Optimal Analysis of the Cube versus Cube Test". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2023, pp. 2761–2776. DOI: 10.1137/1.9781611977554.ch104. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611977554.ch104. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611977554.ch104.

[MZ24] Dor Minzer and Kai Zhe Zheng. "Near Optimal Alphabet-Soundness Tradeoff PCPs". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. STOC 2024. Vancouver, BC, Canada: Association for Computing Machinery, 2024, 15–23. ISBN: 9798400703836. DOI: 10.1145/3618260.3649606. URL: https://doi.org/10.1145/3618260.3649606.

[McE78] Robert J McEliece. "A public-key cryptosystem based on algebraic coding theory". In: *Jet Propulsion Laboratory DSN Progress Report* 42-44 (1978), pp. 114–116. URL: https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[OS09] Raphael Overbeck and Nicolas Sendrier. "Code-based cryptography". In: *Post-Quantum Cryptography*. Ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–145. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_4. URL: https://doi.org/10.1007/978-3-540-88702-7_4.

[PK22] Pavel Panteleev and Gleb Kalachev. "Asymptotically good Quantum and locally testable classical LDPC codes". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, 375–388. ISBN: 9781450392648. DOI: 10.1145/3519935.3520017. URL: https://doi.org/10.1145/3519935.3520017.

[PS94] Alexander Polishchuk and Daniel A. Spielman. "Nearly-linear size holographic proofs". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '94. Montreal, Quebec, Canada: Association for Computing Machinery, 1994, 194–203. ISBN: 0897916638. DOI: 10.1145/195058.195132. URL: https://doi.org/10.1145/195058.195132.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '16. Cambridge, MA, USA: Association for Computing Machinery, 2016, 49–62. ISBN: 9781450341325. DOI: 10.1145/2897518.2897652. URL: https://doi.org/10.1145/2897518.2897652.

[RS97] Ran Raz and Shmuel Safra. "A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, 475–484. ISBN: 0897918886. DOI: 10.1145/258533.258641. URL: https://doi.org/10.1145/258533.258641.

[Ree00] I.S. Reed. "A brief history of the development of error correcting codes". In: *Computers & Mathematics with Applications* 39.11 (2000), pp. 89–93. ISSN: 0898-1221. DOI: https://doi.org/10.1016/S0898-1221(00)00112-7. URL: https://www.sciencedirect.com/science/article/pii/S0898122100001127.

[SS96] Michael Sipser and Daniel A. Spielman. "Expander codes". In: *IEEE Transactions on Information Theory* 42.6 (1996), pp. 1710–1722.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom Generators without the XOR Lemma". In: *J. Comput. Syst. Sci.* 62.2 (2001), 236–266. ISSN: 0022-0000. DOI: 10.1006/jcss.2000.1730. URL: https://doi.org/10.1006/jcss.2000.1730.

[SU05] Ronen Shaltiel and Christopher Umans. "Simple extractors for all min-entropies and a new pseudorandom generator". In: *J. ACM* 52.2 (Mar. 2005), 172–216. ISSN: 0004-5411. DOI: 10.1145/1059513.1059516. URL: https://doi.org/10.1145/1059513.1059516.

[SV06] Nandakishore Santhi and Alexander Vardy. "Minimum Distance of Codes and Their Branching Program Complexity". In: *2006 IEEE International Symposium on Information Theory*. 2006, pp. 1490–1494. DOI: 10.1109/ISIT.2006.262116.

[Sha79] Adi Shamir. "How to share a secret". In: *Commun. ACM* 22.11 (Nov. 1979), 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[Spi95] Daniel A. Spielman. "Linear-Time Encodable and Decodable Error-Correcting Codes". In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '95. Las Vegas, Nevada, USA: Association for Computing Machinery, 1995, 388–397. ISBN: 0897917189. DOI: 10.1145/225058.225165. URL: https://doi.org/10.1145/225058.225165.

[Sud97] Madhu Sudan. "Decoding of Reed Solomon Codes beyond the Error-Correction Bound". In: *J. Complex.* 13.1 (Mar. 1997), 180–193. ISSN: 0885-064X. DOI: 10.1006/jcom.1997.0439. URL: https://doi.org/10.1006/jcom.1997.0439.

[Tre04] Luca Trevisan. *Some Applications of Coding Theory in Computational Complexity*. 2004. URL: https://eccc.weizmann.ac.il/report/2004/043/.

[Uma03] Christopher Umans. "Pseudo-random generators for all hardnesses". In: *J. Comput. Syst. Sci.* 67.2 (2003), 419–440. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(03)00046-1. URL: https://doi.org/10.1016/S0022-0000(03)00046-1.

[Vid15] Michael Viderman. "A combination of testability and decodability by tensor products". In: *Random Struct. Algorithms* 46.3 (May 2015), 572–598. ISSN: 1042-9832. DOI: 10.1002/rsa.20498. URL: https://doi.org/10.1002/rsa.20498.

[Woz58] J.M. Wozencraft. "List decoding". In: *Quterly Progress Report, Research Laboratory of Electronics* 48 (1958), pp. 90–95.

[Yek12] Sergey Yekhanin. *Locally Decodable Codes*. Hanover, MA, USA: Now Publishers Inc., 2012. ISBN: 1601985444.

A Non uniform List Recovery

A.1 List Decoding by Finding a Good Coin in a Pile

If one is only interested in a *non-uniform* algorithm for time-space efficient list decoding, one can use ideas from [GM16] to obtain a simple algorithm that uses a time-space efficient unique decoding algorithm as a blackbox. The algorithm relies on local testing, but local testing for large constant agreement γ suffices. Therefore, this algorithm can be performed for locally testable and correctable codes other than Reed-Muller.

The properties we need from a code for our non-uniform results are:

Locally Testable (Low Error Regime): This is a locally testable code that gives a close multiplicative estimate of the distance to the nearest codeword. This is local testing in the low error (high agreement) regime where words are close to codewords, which is easier than the high error (low agreement) regime needed for our uniform decoders.

Definition A.1 (Locally Testable Codes (Low Error, Informal))**.** *We say that a code $C : \Sigma_1^k \to \Sigma_2^n$ is a locally testable code (LTC) with q queries and approximation factor α if there is a tester that makes q queries to a received word $w \in \Sigma^n$ and either accepts or rejects such that if the tester accepts with probability β, then there is a codeword $C(x)$ that agrees with w on at least β/α and at most $\alpha\beta$ fraction of positions.*

Typical Locally Correctable: This means the local corrector is non-adaptive (where it queries is only a function of the randomness), the queries are smooth (for each index we want to correct, no particular symbol in the input is queried with high probability), and the corrector has perfect completeness (the local corrector always succeeds when the input is a codeword).

Definition A.2 (Typical Locally Correctable Codes)**.** *For any systematic code $C : \Sigma_1^k \to \Sigma_2^n$ we say that C is a typical LCC if it is an LCC and has a corrector that is smooth, non-adaptive, and has perfect completeness.*

Locally List Correctable: This is the standard notion of locally list correctable that first generates a short list of decoders such that with high probability, every nearby codeword is calculated by one of these decoders.

Definition A.3 (Locally list correctable code)**.** *A code $C : \Sigma^k \to \Sigma^n$ is locally list correctable with q queries, ε correction error, δ relative decoding radius, and L list size, if it has a randomized D algorithm that makes q queries to a word $w \in \Sigma^n$ and outputs local correctors $D_1, \ldots, D_L$ such that the following holds. With probability at least $1 - \varepsilon$, for each codeword $C(x)$ that is close to w, that is, $w_j = C(x)_j$ for at least $1 - \delta$ fraction of $i \in [n]$, for some $j \in [L]$, for all $i \in [n]$ the local corrector D_j makes q queries to w, and outputs $C(x)_i$.*

Exact definitions can be seen in Section 4. See Theorem A.10 for a full theorem statement.

Theorem A.4 (Locally List Recoverable and Testable Codes Have Efficient Deterministic Decoding (Informal))**.** *Suppose we have a code $C : \Sigma^k \to \Sigma^n$ that is a locally testable code, typical locally correctable code, and is locally list decodable with list decoding radius d, output list length L, and $n^{o(1)}$ queries. Take any $\xi > 0$ with $\xi = o\left(\frac{1}{\log(L)}\right)$.*

Then there is a deterministic, non-uniform global *list recovering algorithm for C with output list length L, and list recovering radius $d - \xi n$ that runs in time*

$$\tilde{O}\left(n^{1+o(1)} \frac{nL^{O(1)}}{(n-d)\xi^2} \log(|\Sigma|)\right)$$

and space

$$O\left(n^{o(1)} L \log(L) \log(|\Sigma|)\right).$$

The idea is as follows. Consider the different randomness settings for a randomized local corrector. Each of them leads to the correction of some fraction of the indices (possibly 0). Most of them correct most indices. We wish to find one randomness string that leads to the correction of most indices. We can then invoke our unique decoding algorithm on the word to decode the message. To aid our search, we use local tests to estimate the fraction of indices that are corrected for a randomness string.

In the paper [GM16] a setup of this nature was described using the following metaphor: There is a pile of coins. Each coin falls on heads with some probability ("bias"). Suppose that most of the coins in the pile are highly biased. We wish to find one biased coin. We can flip any coin we want.

The analogy is that each randomness string corresponds to a coin. Flipping the coin corresponds to choosing a local test, doing local correction to run that test on the supposedly corrected codeword, and outputting whether the test passes[3]. The bias of the coin is roughly the fraction of indices that are corrected. We wish to find a highly biased coin, namely a randomness string that corrects most indices. The paper [GM16] shows that by making $\tilde{O}(m)$ coin flips one can find a biased coin with probability at least $1 - 2^{-m}$. From this one can get a non-uniform algorithm for list decoding via Adleman's argument: Once the error probability goes below the number of inputs, there must exist randomness for the coin flipping algorithm that works for all inputs. Fix this randomness as non-uniform advice for the algorithm. The corrector may still have some errors, but few enough that we can use time and space efficient unique decoding to remove the remaining errors.

Comparison of uniform and non-uniform results. Our uniform and non-uniform results both utilize local testability to improve beyond the unique decoding radius, but they use them differently. Our non-uniform corrector first generates distinct correctors that output words close to codewords and then these are further corrected independently. In contrast, our uniform decoders remove all errors first (it does codewords list recovery), then it separates them into distinct codewords.

[3]More specifically, there are many decoded words and we check if enough of them pass. See Appendix A.1 for more details.

For our non-uniform corrector we use local testing to make sure that the starting decoders correspond to codewords. This local test only needs to confirm that the number of errors is very low, so it only needs to get a good estimate of the number of errors in the low error regime. These tests are used in conjunction with a non-uniform coin flipping procedure that our uniform correctors do not use. In contrast, our uniform corrector needs local tests that give a good estimate of the error when it is very close to one. This is because our uniform tester is not used directly, instead it is embedded in the local corrector and implies that with high probability there are few out-of-codeword errors after the first round of correction. In both algorithms, after the first round (choosing the close enough decoders for non-uniform case and removing out-of-codeword errors for the uniform case) local testing is no longer used.

Our non-uniform correctors are more efficient than the uniform ones because known uniform local testers and correctors are not as randomness efficient as our non-uniform testers and correctors.

A.2 Challenges in Non-Uniform List Recovery

One issue is that the coin flipping protocol doesn't promise we get a perfect coin (one that has perfect bias and all the tests always pass). It instead only promises we get a coin that has high bias. Said another way, there is a gap between what the protocol is promised exists versus what it actually gives. Informally, this corresponds to local correctors that sometimes fail, or they output codewords that are not quite as close to the input word as we desire. One difficulty this adds is that there is still some error in the codeword, which we address with efficient unique decoding.

Our coin flipping subroutine also has another, more subtle technical hurdle. Namely, we don't know how many codewords are close to the provided word *and* there may be several codewords near the list decoding radius. In contrast, for unique decoding we always know there is exactly one nearby codeword (if there are any at all). The issue is during local testing: how do we know the local list correction is correcting to all the nearby codewords?

One solution is to count the number of nearby codewords, and make sure there are enough. Indeed, we do this in our local test. The challenge is that there may be some codewords near the decoding radius that have an adversarial probability of passing a test. Indeed, the coin flipping protocol is only promised to not return any coin with a bias below some gap. This corresponds to accepting some codewords that are slightly farther than the decoding radius. So if we are only counting the number of nearby codewords, we may count a distant codeword and miss a nearby codeword. This is unacceptable. Our deterministic local corrector must always output every nearby codeword.

The solution is to tailor the local tests so that there are no codewords whose distance lies in the gap of our test. So if we really get local correctors that pass our test with high probability, they must correct to all the nearby codewords since no codeword will be at an adversarial distance to confuse the tester. Of course, we cannot be sure that there will be no codewords within any specific constant distance from the provided word. So we have to try several small gaps and use a counting argument to show one of them must not have a codeword whose distance is in that specific gap. For this counting argument to work, we need to have a bound on the maximum number of codewords within a given distance. This is why we lose a small factor in the correcting radius. There could be many codewords whose distance is just larger than the correcting radius. So our gaps need to all be within the original correcting radius, costing us the ξn factor in our decoding radius in the result.

A.3 Testing Local Correctors

Our test has three parts. One part tests that local correctors output codewords. This comes from local testing. The second is that these codewords are near the input multi-string, and the third is that they are different from each other. The second and third comes from random sampling to estimate their relative agreement.

Lemma A.5 (Testing if Strings Are Close)**.** *For any integers n, d, ℓ, alphabet Σ, and constants $\xi, s > 0$, there is a randomized test V that takes as input a string $w^1 \in \Sigma^n$, and an ℓ-multi-string $w^2 \in \binom{\Sigma}{\ell}^n$ such that:*

Completeness: *If* $\Pr_{i\in[n]}[w_i^1 \notin w_i^2] \leq \frac{d}{n} - \xi$*, then the algorithm accepts with probability at least* $1-s$*. We call* $1-s$ *the completeness.*

Soundness: *If* $\Pr_{i\in[n]}[w_i^1 \notin w_i^2] > \frac{d}{n}$*, then the algorithm accepts with probability at most* s*. We call* s *the soundness.*

Efficiency: *The algorithm makes* $O\left(\frac{n\log(1/s)}{(n-d)\xi^2}\right)$ *queries to* w^1 *and* w^2 *and runs in time* $O\left(\log(n)\frac{n\log(1/s)}{(n-d)\xi^2}\right)$ *(the* $\log(n)$ *factor is just to describe the query locations).*

Proof. The algorithm will make $\frac{16n\log(1/s)}{(n-d)\xi^2}$ queries to the same random places in w^1 and w^2. If at least a $\frac{n-d}{n} + \frac{\xi}{2}$ fraction of the sampled j have $w_j^1 \in w_j^2$, then we accept. The completeness and soundness follows from a Chernoff bound.

For completeness, if $\Delta(w_1, w_2) \leq d - n\xi$, then the expected number of points w_1 and w_2 will agree on is $\mu \geq \frac{16n\log(1/s)}{(n-d)\xi^2}\left(\frac{n-d}{n} + \xi\right) \geq 16\frac{\log(1/s)}{\xi^2}$. Then by a Chernoff bound, the probability that our sample has $\xi/2$ fraction fewer points they agree is at most

$$\begin{aligned} e^{-(\xi/2)^2\mu/2} \leq & e^{-\xi^2 16\frac{\log(1/s)}{\xi^2}/8} \\ < & s. \end{aligned}$$

A similar argument holds holds for completeness. □

Now we give a test for checking how many codewords near a given multi-string w are actually (nearly) described by $w^1, \ldots, w^L$.

Lemma A.6 (Testing Claimed Codewords)**.** *Take any locally testable code* $C : \Sigma_1^k \to \Sigma_2^n$ *with distance* d_0 *approximation factor* α *and* q_t *testing queries. Now suppose one is given as input strings* $w^1, \ldots, w^L \in \Sigma_2^n$*, integers* $\ell, L' \leq L$*, integers* d_1, d_2*, and an* ℓ*-multi-string* $w \in \binom{\Sigma_2}{\ell}$*. Then there is a test,* V*, making queries to* w *and each* w^j *such that:*

Completeness: *If*

$$\left|\{x \in \Sigma_1^k : \exists j \in [L], w^j = C(x) \wedge \Pr_{i\in[n]}[w_i^j \notin w_i] \leq \frac{d_1 - d_2}{n} - \xi\}\right| \geq L'$$

then the test passes with probability at least $\frac{3}{4}$*.*

Soundness: *If*

$$\left|\{x \in \Sigma_1^k : \exists j \in [L], \Delta(C(x), w^j) \leq d_2 \wedge \Pr_{i\in[n]}[C(x)_i \notin w_i] \leq \frac{d_1}{n}\}\right| < L'$$

and $d_2 \leq \frac{d_0}{16L(\log(L)+2)}$ *then the test passes with probability at most* $\frac{1}{4}$*.*

Efficient: *If the local tester for the code runs in time* T *and space* S*, this new tester runs in time*

$$O\left(L\log(L)\frac{n}{d_2}\left(T\alpha + \frac{\log(L)d_2}{d_0} + \frac{d_2\log(n)}{(n-d_1+d_2)\xi^2}\right)\right)$$

and space

$$O\left(S + \log(n) + L\frac{n}{d_0}\log(L)\right)$$

while making

$$O\left(L\log(L)\frac{n}{d_2}\left(q_t\alpha + \frac{d_2}{(n-d_1+d_2)\xi^2}\right)\right)$$

queries to $w^1, \ldots, w^L$*.*

Informally, the completeness condition says that if there are L' codewords among $w^1, \ldots, w^L$ that are *also* near w, then we will accept with probability 3/4. The soundness condition says that if there are not L' codewords even close to any of the $w^1, \ldots, w^L$ that are *also* close to w, then the test will reject with probability 3/4. We think of $n > d_0 > d_1 > d_2$ and $d_2 = n^{1-\epsilon}$ for a very small ϵ so that the test runs in time close to n^ϵ.

Proof. Our test proceeds in three parts. The first part tests the strings $w^1, \ldots, w^L$ to see which are codewords. Our second test makes sure these codewords are near w. Our third test verifies that the strings are *different* codewords. Now we will state the tests again in more detail.

The first test runs the local tester $\frac{\alpha n}{d_2}(\log(L)+3)$ times for each string and a string passes if all tests pass. The second test compares $O\left(\frac{n\log(L)}{(n-d_1+d_2)\xi^2}\right)$ random locations of each w^j to w (using Lemma A.5 with soundness $\frac{1}{16L}$) to check if each of the w^j that pass the first test are within $d_1 - d_2 - n\xi$ of w, or further than $d_1 - d_2$ from w. We use $d_1 - d_2$ because w^j may be d_2 from $C(x)$ and we want to know if $C(x)$ is within d_1 of w. For the third test we choose $\frac{n}{d_0}(2\log(L)+2)$ random locations and the collection of strings pass if there at least L' different strings passing the first two tests that differ on those random locations.

Now we verify the test is as claimed.

Completeness: Let $W_c = \{C(x) : x \in \Sigma_1^k, \exists j \in [L], w^j = C(x) \wedge \Pr_{i\in[n]}[w_i^j \notin w_i] \leq \frac{d_1-d_2}{n} - \xi\}$. Suppose that $|W_c| \geq L'$. First see that all $w \in W_c$ pass the first test. By the completeness of Lemma A.5 and a union bound, the probability that any $w \in W_c$ fails the second test is at most $\frac{1}{16}$. By the distance of C, we also see that for every pair of strings in W_c, the probability that they agree on a random index is at most $1 - \frac{d_0}{n}$. So the probability they agree on all $\frac{n}{d_0}(2\log(L)+2)$ locations is at most

$$\begin{aligned}\left(1-\frac{d_0}{n}\right)^{\frac{n}{d_0}(2\log(L)+2)} &\leq e^{-2\log(L)-2}\\ &\leq \frac{1}{7L^2}.\end{aligned}$$

So by a union bound, the probability that any pairs of strings of W agree on the random points is at most $\frac{1}{7}$. Thus with probability at most $\frac{1}{4}$ will the test fail.

Soundness: Let $W_s = \{C(x) : x \in \Sigma_1^k, \exists j \in [L], \Delta(C(x), w^j) \leq d_2 \wedge \Pr_{i\in[n]}[C(x)_i \notin w_i] \leq \frac{d_1}{n}\}$. Now suppose that $|W_s| < L'$. First we will argue that with low probability will any string far from every string in W_s pass. Then we will argue that with low probability will at least L' strings close to W_s disagree on the random locations.

Consider any w^j that is not within distance d_2 of any string in W_s. If $\Delta(w^j, C) > d_2$, since C is locally testable, the probability a local test fails is at least $\frac{d_2}{\alpha n}$. Thus the probability that w^j passes all $\frac{\alpha n}{d_2}(\log(L)+3)$ local tests is at most

$$\begin{aligned}\left(1-\frac{d_2}{\alpha n}\right)^{\frac{\alpha n}{d_2}(\log(L)+3)} &\leq e^{-\log(L)-3}\\ &\leq \frac{1}{8L}.\end{aligned}$$

Suppose there is any $x \in \Sigma_1^k$ such that $\Delta(w^j, C(x)) \leq d_2$. If $\Pr_{i\in[n]}[w_i^j \notin w_i] < \frac{d_1-d_2}{n}$, then $\Pr_{i\in[n]}[C(x)_i \notin w_i] \leq \Pr_{i\in[n]}[w_i^j \notin w_i] + \frac{\Delta(w^j, C(x))}{n} \leq \frac{d_1}{n}$, which would imply $C(x) \in W_s$, a contradiction since w^j is far from W_s. Thus $\Pr_{i\in[n]}[w_i^j \notin w_i] \geq \frac{d_1-d_2}{n}$ and by the soundness of Lemma A.5, with probability at most $\frac{1}{16L}$ will the second test pass. So by a union bound, with probability at most $\frac{1}{8}$ would any string d_2 far from every codeword within d_1 of w pass the first two tests.

Now consider any w^j such that for some $w' \in W_s$ we have that $\Delta(w^j, w') \leq d_2$. Then we will show with high probability w and w^j will agree on the random points chosen. The probability they disagree on a random point is at most $\frac{d_2}{n}$. So by a union bound, the probability they disagree at any of the $\frac{n}{d_0}(2\log(L)+2)$ points is at most $\frac{2d_2}{d_0}(\log(L)+1)$. So by a union bound, the probability any w^j within

d_2 of a $w' \in W_s$ disagrees with that w' is at most $\frac{2d_2}{d_0}(\log(L)+1)L$. In particular, if $d_2 \leq \frac{d_0}{16L(\log(L)+1)}$, then the probability is at most $\frac{1}{8}$.

See that if only strings within d_2 of a codeword in W_s pass the first two tests, and every string within d_2 of a codeword in W_s agrees with some string in W_s on the randomly chosen points, then the test will fail since $|W_s| < L'$. So by a union bound, the probability the test passes is at most $\frac{1}{4}$.

Efficient: The number of queries for the first test is L times the number of queries for the local tester times the number of local tests. This is $L\frac{\alpha n}{d_2}(\log(L)+3)q_t$. The number of queries for the second test is $O\left(L\frac{n\log(L)}{(n-d_1+d_2)\xi^2}\right)$. The number of queries for the third test is L times the number of locations queried, which is $L\frac{n}{d_0}(2\log(L)+2)$. So the total number of queries is:

$$\begin{aligned}
&L\frac{\alpha n}{d_2}(\log(L)+3)q_t + O\left(L\frac{n\log(L)}{(n-d_1+d_2)\xi^2}\right) + L\frac{n}{d_0}(2\log(L)+2)\\
=&O\left(L\log(L)\left(\frac{\alpha n}{d_2}q_t + \frac{n}{(n-d_1+d_2)\xi^2} + \frac{n}{d_0}\right)\right)\\
=&O\left(L\log(L)\frac{n}{d_2}\left(\alpha q_t + \frac{d_2}{(n-d_1+d_2)\xi^2}\right)\right).
\end{aligned}$$

The time is similar, except that we also need to count the number collisions when restricting the strings to random locations. To do this, we sort the L strings restricted to these random locations. This only adds an extra $O(L\log(L)\frac{n}{d_0}\log(L))$ time steps.

□

A.4 Finding Good Enough Local Correctors

Once we can test if a given set of local correctors corrects most errors for a given multi-string, we just need to choose local correctors that pass with high probability. If it passes the test with high probability, then it will output local correctors that correct to strings near the codewords we actually want. At that point we can use unique decoding to finish the job.

To complete our argument, we need a subroutine for solving the coin flipping problem by Grossman and Moshkovitz [GM16, Lemma 4.2]. This lets us efficiently find a local corrector that is heavily biased (or at least, biased above $\frac{1}{4}$). We give the specific instantiation of this algorithm for our setting. This algorithm takes a randomized algorithm, and a test for that randomized algorithm, and finds a good output with very high probability. Using this, we efficiently find local correctors that are good enough with such high probability that some choice of randomness always succeeds.

Lemma A.7 (Finding A Good Coin With High Probability)**.** *Suppose that there is a randomized algorithm, A, that produces an output, B. Suppose that there is a randomized algorithm V that takes B as input and either accepts or rejects B. Suppose that both A and V both run in time T and space S, and that $|B| = O(S)$. Finally, suppose that*

$$\Pr_B[\Pr[V \text{ accepts } B] < 1-\eta] < 1/2.$$

Then for any constant $\xi > 0$ and integer n, there is an algorithm that runs in time

$$O\left(nT\left(\frac{\log(n/\xi)}{\xi}\right)^2\right)$$

and space

$$O(S + \log(n/\xi))$$

and with probability 2^{-n} outputs a B such that V accepts with probability at least $1-\eta-\xi$.

In particular if $\xi = \Omega(1)$, then the time is $O(n\log(n)^2 T)$ and the space is $O(S + \log(n))$.

Now combining these results, we can get a randomized algorithm for list correctable codes that can find a list of L' good deterministic local correctors with very high probability (if there are L'). Of course, by good, we mean the local correctors output strings close to the nearby codewords (think within $n^{1-\epsilon}$).

We note that this test only promises we find L' codewords within $\frac{d_1}{n}$ of w if there are L' codewords within $\frac{d_1-d_2}{n} - \xi$ of w. It does not promise that it will return the L' codewords that are specifically within $\frac{d_1-d_2}{n} - \xi$. So if there is a codeword whose distance from w is between $\frac{d_1}{n}$ and $\frac{d_1-d_2}{n} - \xi$, we may return that nearby codeword and not a closer one.

Lemma A.8 (Finding Good List Of Deterministic Correctors)**.** *Take any locally testable code $C : \Sigma_1^k \to \Sigma_2^n$ with distance d_0 approximation factor α and q_t testing queries whose local tester runs in time T_t and space S_t. Suppose that C is also locally list recoverable with q_c list correcting queries, list correcting soundness $\frac{1}{8Ln}$, list correcting radius d_1, input list length ℓ and output list length L whose local list correcting algorithm runs in time T_c and space S_c.*

There is a randomized algorithm, D', that takes as input a constant $\xi > 0$, integers d_2, L', and m where $d_2 \leq \frac{d_0}{16L(\log(L)+2)}$ and $d_2 \leq d_1 - d_2 - \xi n \leq d_1 \leq d_0$ and a multi-string $w \in \binom{\Sigma_2}{\ell}^n$ that either rejects or outputs $O(T_c)$ bits describing a list of L different deterministic algorithms, $D'_1, \ldots, D'_L$ which have oracle access to w and take an index $j \in [n]$ and outputs a symbol in Σ_2. Then the following holds:

Efficient: *D' runs in time*

$$O\left(m\log(m)^2 L\log(L)\frac{n}{d_2}\left(T_c\left(\alpha q_t + \frac{d_2}{(n-d_1)\xi^2}\right) + \frac{\log(L)d_2}{d_0} + \frac{d_2\log(n)}{(n-d_1)\xi^2} + \alpha T_t\right)\right)$$

and space

$$O\left(S_t + S_c + \log(mn\alpha L) + T_c + L\frac{n}{d_0}\log(L)\right).$$

Further, each D'_i runs in time T_c and space S_c and makes only q_c queries to its input.

Soundness: *If $|\{C(x) : x \in \Sigma_1^k, \Delta(C(x), w) \leq d_1\}| < L'$, then with probability at least $1 - 2^{-m}$ will D' reject.*

Completeness: *For $j \in [L]$, define $w'^j \in \Sigma_2^n$ by $w'^j_i = D'^w_j(i)$. If*

$$\left|\{x \in \Sigma_1^k : \Pr_{i\in[n]}[C(x)_i \notin w_i] \leq \frac{d_1-d_2}{n} - \xi\}\right| \geq L',$$

then with probability at least $1 - 2^{-m}$ will D' output $D'_1, \ldots, D'_L$ such that

$$\left|\{x \in \Sigma_1^k : \Pr_{i\in[n]}[C(x)_i \notin w_i] \leq \frac{d_1}{n} \wedge \exists j \in [L], \Delta(w'^j, C(x)) \leq d_2\}\right| \geq L'.$$

Proof. The output deterministic algorithms are just the local correctors from the local list decoder for C with the randomness hard coded. The only trick is to get a good answer with higher probability, we need to test the local decoders we get. To do this, we view the test of Lemma A.6 as a test of how good the deterministic correctors are, then we run Lemma A.7 to efficiently find a good local decoder. Finally, we run the test of Lemma A.6 $10m$ time mores on the decoders that are chosen by the coin flipping subroutine and reject if more than half the tests fail.

Let V be the test of Lemma A.6. If there are less than L' codewords within d_1 of w, then by the soundness of the test, for any deterministic decoders that are given, V will accept with probability at most $\frac{1}{4}$. So by a Chernoff bound, if there are $10m + 10$ tests run, the probability more than half the tests pass is at most 2^{-m-1}. So the soundness holds.

If there are more than L' codewords within $d_1 - d_2 - \xi n$ of w, then with probability at least $2/3$ will the randomized decoders chosen by the list recovery algorithm, $D_1, \ldots, D_L$, contain decoders that decode to these L' closest codewords with probability $1 - \frac{1}{8nL'}$. By a union bound, if their randomness is hard coded, chosen uniformly at random, the probability that $D_1, \ldots, D_L$ contain decoders that decode to these

L' closest codewords is at least $\frac{1}{2}$. By completeness of Lemma A.6, such a set of decoders passes the test with probability at least $\frac{3}{4}$. By Lemma A.7, the probability the coin flipping protocol chooses decoders that don't accept with probability at least $\frac{1}{2}$ is at most 2^{-m-1}. By soundness of Lemma A.6, if the test passes with probability at least $\frac{1}{2}$, then there must be L' codewords within d_1 of w that $D_1, \ldots, D_L$ decode to within distance d_2. So the probability that the chosen $D'_1, \ldots, D'_L$ do not decode to strings within d_2 of L' codewords within d_1 of w is at most 2^{-m-1}.

Again by a Chernoff bound, the probability the final $10m + 10$ tests will fail over half of the attempts is at most 2^{-m-1}. Thus the probability that the test either chooses an invalid $D'_1, \ldots, D'_L$ or rejects is at most 2^{-m}. So completeness holds.

For the efficiency, see that we one only needs to run the the local test and local correction for each round of the coin flip subroutine. So the time is the time to run $O(m \log(m))$ local tests plus the time to make all the queries for those tests times the time to make all those queries. □

A.5 Completing The Local Correctors

Once we have local correctors that output strings near the codewords, we can do unique decoding on them. This lemma from [CM25] shows that an efficient global decoder for unique decoding exists.

Lemma A.9 (Typical LCCs have Efficient Deterministic Correctors)**.** *Suppose code $C : \Sigma_1^k \to \Sigma_2^n$ is a typical LCC with smoothness β, and q queries as well as an LCC with correcting radius d'_2 and q queries.*

Then the code C has a deterministic non-uniform corrector and decoder with correcting and decoding radius $d_2 = d'_2/3$ running in time

$$O\left(nq^2\left(\beta + 2^{O\left(\sqrt{\log(n/\beta)\log(q)}\right)}\right)\log(|\Sigma_2|)\right)$$

and space

$$O\left(q\left(\log(\beta) + \log(n)\sqrt{\frac{\log(n/\beta)}{\log(q)}}\right)\log(|\Sigma_2|\right).$$

Now applying this unique decoder with our algorithm for choosing local decoders that output a string near a codeword, we get our efficient list recovery algorithm.

More specifically, our local corrector does the following. For each $i \in [L]$, we run the coin flipping procedure with a test who looks for codewords within $d_1 - \xi i/L$ of the provided input with the test's gap set to $\xi/2L$. We run this for every $L' \in [L]$ to find as many codewords within $d_1 - \xi i/L$ as possible. For some two successive intervals, we will find the same number of nearby codewords. This means that the closer correctors must have found all codewords near the input. Use the codewords from that closer decoding radius.

This will give approximations of all the codewords within $d_1 - \xi$. Finally, we run unique decoding on these codewords to remove any errors in our approximations.

Theorem A.10 (Locally List Recoverable and Testable Codes Have Efficient Deterministic Decoding)**.** *Suppose we have a code $C : \Sigma_1^k \to \Sigma_2^n$ with distance d_0 that is a:*

LTC *with approximation factor α with q_t queries.*

Typical LCC *with unique correcting radius d'_2, smoothness β, correcting soundness $\frac{1}{11n}$, and q_c queries.*

Locally List Recoverable *with list recovering radius d_1, input list length ℓ, output list length L, list correcting soundness $\frac{1}{8Ln}$, and q_c queries.*

Then for any $\xi > 0$ such that $d'_2 > \frac{3\xi n}{2(L+1)}$ and $\xi \leq \frac{d_0}{8n(\log(L)+2)}$, there is a deterministic, non-uniform global *list recovering algorithm for C with input list length ℓ, output list length L, and list recovering radius $d_1 - \xi n$ that runs in time*

$$O\left(n\left(\frac{n\log(n)^2}{n-d_1}\frac{L^4\log(L)^2 q_c q_t \alpha}{\xi^2} + Lq_c^3\left(\beta + 2^{O\left(\sqrt{\log(n/\beta)\log(q_c)}\right)}\right)\right)\log(|\Sigma_2|)\right)$$

and space

$$O\left(\left(q_t+\log(nL)+L\frac{n}{d_0}\log(L)+q_c\left(\log(\beta)+\log(n)\sqrt{\frac{\log(n/\beta)}{\log(q_c)}}\right)\right)\log(|\Sigma_2|)\right).$$

Proof. To get the final non uniform algorithm, the basic idea is to run the protocol from Lemma A.8 with $m=n+1$ for $\log(L)$ times to figure out how many codewords are near w. In particular, we want to run it with $d_2=\frac{\xi n}{2(L+1)}$ and its ξ' set to $\frac{\xi}{2(L+1)}$. Unfortunately, the algorithm of Lemma A.8 is only guaranteed to find as many strings within d_1 of w as there are within $d_1-\frac{\xi n}{L+1}$. So it is quite possible that it gives a corrector to a codeword farther than $d_1-\frac{\xi n}{L+1}$ and misses a codeword much closer to w, perhaps even the closest codeword. To get around this, we will run Lemma A.8 for L' more times to get the codewords closer than $d_1-\frac{\xi n}{L+1}, d_1-\frac{2\xi n}{L+1},\ldots,d_1-\frac{L\xi n}{L+1}$. In one of the intervals $[d_1-\frac{i\xi n}{L+1}, d_1-\frac{(i-1)\xi n}{L+1}]$, there must not be any codewords whose distance from w is in that interval. So our corrector for that level correctly finds all the nearby codewords. Finally, we can run our efficient deterministic decoding on our list of deterministic decoders we previously selected to correct our codeword.

Specifically, setting $d_2=\frac{\xi n}{2(L+1)}$, see that for any $i\in[L+1]$, and any proposed setting of L', we can run Lemma A.8 with $m=n+1$ to find local decoders for d_2 approximations of codewords within $d_1-\frac{(i-1)\xi n}{L+1}$ of w if there are L' codewords within $d_1-\frac{i\xi n}{L+1}$ of w with probability $1-2^{-n-1}$, and if there aren't L' codewords within $d_1-\frac{i\xi n}{L+1}$ of w, then we reject with probability at least $1-2^{-n-1}$. Thus some choice of randomness will always succeed and we hard code that randomness, so we have a deterministic procedure that always works. For each $i\in[L+1]$, by using a binary search on L' we can find an L'_i such that there are at most L'_i codewords within $d_1-\frac{i\xi n}{L+1}$ of w, and at least L'_i codewords within $d_1-\frac{(i-1)\xi n}{L+1}$. Since there are only L codewords within d_1 of w, we must have for some i that $L'_i=L'_{i+1}$. In particular, this means that the local correctors for the closer distance, L'_{i+1}, must have correct to all the codewords within $d_1-\frac{(i-1)\xi n}{L+1}$, since it corrects to L'_i of them and there are only L'_i codewords within $L'_{i+1}=L'_i$ of them. So take these local decoders, and apply Lemma A.9.

For Lemma A.8 to succeed, we need $d_2\le\frac{d_0}{16L(\log(L)+2)}$, but this holds since $d_2=\frac{\xi n}{2(L+1)}$ and $\xi\le\frac{d_0}{8n(\log(L)+2)}$, so this protocol succeeds in finding the local correctors. That is, we give deterministic local correctors $D'_1,\ldots,D'_L$ such that for every codeword within $d_1-\xi n$, there is some D'_i that outputs a d_2 approximation of it. Since $d'_2>\frac{3\xi n}{2(L+1)}\ge 3d_2$, we also have that Lemma A.9 succeeds for each D'_i correcting to such a codeword. Thus the local recovery succeeds.

Now we analyze the runtime. To find the initial local correctors, we need to run Lemma A.8 $(L+1)\log(L)$ times with its ξ set to $\xi'=\frac{\xi}{2L+1}$ and its d_2 set to $d_2=\frac{\xi n}{2(L+1)}$. We also note that since we are in the non-uniform setting, we can upper bound the time and space of the algorithm by the number of queries times $\log(|\Sigma_2|)$. So running Lemma A.8 requires time

$$\begin{aligned}
&O\left(L\log(L)\left(m\log(m)^2L\log(L)\frac{n}{d_2}\left(T_c\left(\alpha q_t+\frac{d_2}{(n-d_1)\xi'^2}\right)+\frac{\log(L)d_2}{d_0}+\frac{d_2\log(n)}{(n-d_1)\xi'^2}+\alpha T_t\right)\right)\right)\\
\le&O\left(n\log(n)^2L^2\log(L)^2\frac{L}{\xi}\left(q_c\left(\alpha q_t+\frac{Ln}{(n-d_1)\xi}\right)+\frac{1}{L}+\frac{n\log(n)L}{(n-d_1)\xi}+\alpha q_t\right)\log(|\Sigma_2|)\right)\\
\le&O\left(n\frac{n\log(n)^2}{n-d_1}\frac{L^4\log(L)^2}{\xi^2}q_cq_t\alpha\log(|\Sigma_2|)\right)
\end{aligned}$$

and space

$$\begin{aligned}
&O(S_t+S_c+\log(mn\alpha L)+T_c+L\frac{n}{d_0}\log(L))\\
\le&O\left((q_t+q_c)\log(|\Sigma_2|)+\log(nL)+L\frac{n}{d_0}\log(L)\right).
\end{aligned}$$

Then running the final L local correctors of Lemma A.9 requires time

$$\begin{aligned} &O\left(Lq_c n q_c^2 \left(\beta + 2^{O\left(\sqrt{\log(n/\beta)\log(q_c)}\right)}\right)\log(|\Sigma_2|)\right) \\ =&O\left(nLq_c^3 \left(\beta + 2^{O\left(\sqrt{\log(n/\beta)\log(q_c)}\right)}\right)\log(|\Sigma_2|)\right) \end{aligned}$$

and space

$$O\left(\log(L) + q_c\left(\log(\beta) + \log(n)\sqrt{\frac{\log(n/\beta)}{\log(q_c)}}\right)\log(|\Sigma_2|)\right).$$

Putting these together gives the final stated time and space. □

Remark (Efficient Soft Decoding)**.** *We note that while we state all the results for the list recovery, the more general result for soft decoding also holds. Similarly, list decoding is a special case of list recovery where* $\ell = 1$.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il