Time and Space Efficient Deterministic List Decoding

Joshua Cook
Amazon, Seattle, WA, USA

Dana Moshkovitz
Department of Computer Science, The University of Texas at Austin, TX, USA

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 containing exactly the symbols from L codewords, where a small fraction of the lists is corrupted. The goal is to find 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.

2012 ACM Subject Classification Theory of computation → Error-correcting codes; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Reed-Muller code, local correction, local testing

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.42

Related Version *Full version (preliminary)*: `https://eccc.weizmann.ac.il/report/2025/132/`

Funding *Joshua Cook*: This material is based upon work done while at the University of Texas at Austin and supported by the National Science Foundation under grant number 2200956.
Dana Moshkovitz: This material is based upon work supported by the National Science Foundation under grant numbers 2200956 and 2312573.

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 [27, 39]. In

theoretical computer science, codes have numerous applications, e.g., for pseudorandomness [30, 47, 49, 20], interactive protocols [33, 21, 40], probabilistically checkable proofs [5, 18, 3, 2] and cryptography [48, 34, 12, 38, 43].

▶ **Definition 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* [17, 50]. 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 [47, 42], low error PCPs [37, 13, 29, 35] and optimal inapproximability [28].

▶ **Definition 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 $C(x)$ for $x \in \Sigma^k$ with γn agreement with 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 [45, 44]. List decoding too can be achieved in linear time [26]. 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., [45]) were shown to also have decoding in logarithmic space, however the algorithms use large polynomial time. For instance, Spielman codes [45] 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 [7] and [22], respectively. Much of the initial work in the area focused on negative results. Bazzi, Mahdian and Spielman [7] 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 [6] proved that codes that can be encoded in *linear* time and sub-linear space cannot be asymptotically good. Gronemeier [22] 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 [10] constructs explicit asymptotically good codes with

deterministic almost-linear time and sub-linear space encoders [10]. It evades the impossibility result by considering almost linear time instead of linear time. The paper [11] 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 [11] 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 3** (Good Codes With Time-Space Efficient Uniform List Decoding (Informal; See Section 6)). *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. See the end of Section 6 for more detailed results. 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 [23] that reduces list decoding to unique decoding.

1.2 Overview of Our List Decoding Algorithm

Our main technical contribution is generalizing the unique decoding algorithm from the paper [11] to the list decoding setting. The method of [11] 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 [11]. 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.

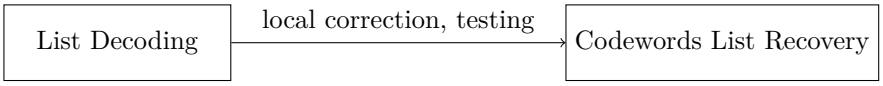

Figure 1 Codewords list recovery is solved in efficient time-space via an iterative correction algorithm (technique from [11]; suitably generalized). List decoding is solved via a time-space efficient reduction to codewords list recovery. Both the algorithm and the reduction use local correction. The 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 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 [11] 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 [11] on unique decoding relies on local correction. The codewords in the paper [10] or in Turbo-like codes [7] 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 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 [11] indeed have low weight constraints. The codes considered in [10, 7] are not locally correctable but they have low weight constraints too. Since self dual codes do not have low weight constraints, this recovers the result of [41] that self dual codes cannot be encoded time and space efficiently and generalizes it to decoding, and more general codes.

One can add a local constraint to any code (e.g., by repeating a symbol in the codeword) and foil the premise of Theorem 5. However, the theorem rules out many natural 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 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 correspond to evaluations 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 $ST = \Omega(d(\mathsf{deg} + 1 - S))$.*

The theorem also suggests a path for ruling out more contrived codes from having time-space efficient encoding or decoding: Find a sub-code without low weight constraints that still has time-space efficient encoding or decoding.

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 [11]. 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 [31, 51]. 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 7** (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 [11] succeeds in derandomizing (global) decoding without significantly increasing the runtime. Unfortunately, the derandomization requires the unique decoding setup.

The paper [11] 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 8** (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 [11] 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 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 [11] from $\ell = 1$ to general ℓ. The algorithm is described in detail in Section 4.

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 26). As in the unique decoding case [11], 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 9** (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 $L_i \subseteq \Sigma$, $|L_i| \leq \ell$, 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 [51] 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 [4, 47], 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 [46, 24, 19] 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 $w(x) \notin \{Q^1(x), \ldots, Q^\ell(x)\}$ where $Q^1, \ldots, Q^\ell$ is the list decoding of w, a random line will rarely contain enough of those to output a symbol inconsistent with $Q^1, \ldots, Q^\ell$.

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

▶ **Definition 10** (Sampling). *A family of samples $\{S\}$ where each sample S is a size-s subset of $\mathbb{F}^{dim}$, is a sampler with accuracy error ϵ and strong confidence error δ if for any $A \subseteq \mathbb{F}^{dim}$, $\mu = |A|/|\mathbb{F}^{dim}|$, for at least $1 - \delta\mu$ fraction of the samples S we have*

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

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 [11], 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 [11] 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 4.2 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 [32, 1, 25].

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 [11].

In this subsection 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 uses randomness to make a small number of queries to the word. The tester *accepts* if there exists a codeword that is consistent with the outcome of its queries. Otherwise, the tester *rejects*. This way, queries to codewords are always accepted. In contrast, words that are far from the code should be rejected with good probability. The definition we use asks that the rejection probability of the tester equals the distance of the word from a codeword up to a small *additive* term (equivalently, the acceptance probability of the tester equals the agreement of the word with a codeword up to a small additive term):

▶ **Definition 11** (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 accepts iff the outcome of the queries is consistent with a codeword in C. For any $0 \leq \beta \leq 1$, if the tester accepts with probability β, then there is a codeword 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. In many definitions, 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 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 11 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, namely replacing lines with certain $O(1)$-dimensional subspaces, makes the corrector into a local tester with a small additive approximation $|\mathbb{F}|^{-\Omega(1)}$ [36]. This means that, except with probability $|\mathbb{F}|^{-\Omega(1)}$, whenever a word agrees locally (on a random $O(1)$-dimensional subspace) with a low degree polynomial, this low degree polynomial is consistent with one of the global low degree polynomials close to the word. In other words, there cannot be more than $|\mathbb{F}|^{-\Omega(1)}$ fraction of the $O(1)$-dimensional subspaces on which the word agrees with a low degree polynomial, without the local low degree polynomial agreeing with one of the global low degree polynomials that are close to the word.

At the time of writing this paper, we do not know of other codes with a randomness-efficient local corrector and a small additive approximation. This is the reason our result is only for Reed-Muller codes. 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 $Q^1, \ldots, Q^\ell$ of degree at most deg that agree with w on at least γ fraction of the points in $\mathbb{F}^{\mathsf{dim}}$. In codewords list recovery, we are given a list of sets of symbols, call it W, such that for most $x \in \mathbb{F}^{\mathsf{dim}}$ we have that $W(x) = Q(x)$, where $Q(x) = \{Q^1(x), \ldots, Q^\ell(x)\}$, and we want to find $Q^1, \ldots, Q^\ell$. We wish to construct a codewords list recovery instance that corresponds to the list decoding of w.

Recall that local list correction as in subsection 2.3 converts an input word w to a list of sets W. This algorithm falls short of a reduction due to two issues: (1) It does not guarantee a precise equality $W(x) = Q(x)$ for most $x \in \mathbb{F}^{\mathsf{dim}}$. (2) It is randomized. In the remainder of this subsection we explain how to resolve issue (1). Issue (2) is resolved thanks to average-case local correction. If there are only $n^{o(1)}$ randomness strings per x, our algorithm can try all of them and pick the majority outcome. Note that here we use the fact there is only one correct outcome per x, namely the list $Q(x)$.

Towards resolving issue (1), note that there are two ways that we could have $W(x) \neq Q(x)$. One is that $Q(x) \not\subseteq W(x)$, and the other is that $W(x) \not\subseteq Q(x)$. That is, the first way is that $W(x)$ could be missing symbols from $Q(x)$; we call these *in-codeword errors*. The other is $W(x)$ could contain extra symbols not in $Q(x)$; We call these *out-of-codeword errors*. In-codeword errors mean that there is a global codeword that is close to w but is missed in the local corrector sample. This is a sampling error, and it is rare as discussed in Subsection 2.3. In contrast, out-of-codeword errors correspond to symbols that appear in the local sample of the corrector, but not in any global codeword that is close to w. Our key idea is that out-of-codeword errors are rare if the local corrector is also a *local tester*. This follows from the definition of a local tester discussed in Subsection 2.4. Local testing precludes a situation where many of the local polynomials the corrector sees do not correspond to any global codeword that is close to w. As explained in Subsection 2.4, we adapt our local corrector to incorporate local testing by querying certain $O(1)$-dimensional subspaces instead of lines. Importantly, we only use subspaces for the reduction to codewords list recovery. Everywhere else in this work, local 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 5 that shows that time-space efficient linear codes must have low weight constraints. The general result (see Theorem 17) 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. For linear codes, this is equivalent to the code having dual distance t. 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 12** (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 13** (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 14** (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 15** (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 Locality Theorem

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 16 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 16 many times in Theorem 17 to give the final result.

▶ **Lemma 16** (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 section.

▶ **Theorem 17** (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 16, 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

$$T \geq \Omega\left((t-S)\left(\frac{d}{S}\right)\right) \quad ST \geq \Omega\left((t-S)d\right). \qquad \blacktriangleleft$$

4 Algorithm For Codewords List Recovery

We start by giving a time-space efficient algorithm for codewords list recovery, extending the unique decoding case of [11] to lists. We refer to the input to a list recovery problem as a *multi-string*, where each index has a set of symbols rather than a single symbol.

4.1 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 [11] 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 18** (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*

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

we have that

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

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

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

we have that

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

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 19** (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

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

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

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

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 20** (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*

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

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 19 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 19 chooses a random $f' \in \mathcal{I}$ and compares $f(w)$ to $f'(w)$. See that

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

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 19 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 21** (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*

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

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 20 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 20 directly. In the general case, we get D_{i+1} by running Lemma 20 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. ◀

4.2 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 [11] 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 22** (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 23** (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 [11, 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 24** (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 $|\mathbb{F}|$. Furthermore, $|\mathcal{L}| = |\mathbb{F}|^{a(\mathsf{dim}+2a+O(1))}\mathsf{poly}(\mathsf{dim})$ and $\mathcal{L}$ is time and space $\mathsf{poly}(|\mathbb{F}|^a\mathsf{dim})$ uniform.

In prior work, Cook and Moshkovitz [11] showed that correcting line sets give improving sets for Reed-Muller codes. We show that they further give list improving sets.

▶ **Lemma 25** (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-*T'*, space-*S' *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*, where* $T' = c\left(T + O\left(|\mathbb{F}|\left(\mathsf{dimpolylog}(|\mathbb{F}|) + \mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)\right)\right)\right)$ *and* $S' = S + O((\mathsf{dim}+c)\log(|\mathbb{F}|)) + |\mathbb{F}|\mathsf{poly}\left(\ell\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathsf{deg}}\right)$*. 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*.*

4.3 List Improving Sets From Correcting Lines

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 it is 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 26** (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 symbol at x'. ◀

4.4 Codewords List Recovery For Reed-Muller Codes

Now, we can use our correcting lines from Lemma 24 with Lemma 25 to get a list improving set. We can use that list improving set with Lemma 21 to recover the codeword multi-string. Finally we use Lemma 26 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 6 we combine Lemma 27 with the results in Section 5 to get list decoding in the *high error* regime.

▶ **Lemma 27** (Codewords 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*

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

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

Further, D makes $n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$ queries, runs in time $n^{1+4/a}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim}\ell)$, and space $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$. 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 $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}\ell)$.

5 Reduction From List Decoding to Codewords List Recovery

Now we reduce from list decoding to codewords list recovery, we need a randomness efficient low degree test for Reed-Muller codes. The structure of the test needs to guarantee that we can do local list recovery if a string is near a codeword, and the soundness of the test needs to guarantee that we do not often decode extra symbols that are not from a nearby codeword.

5.1 Low Degree Testing

The local tester we use is the space vs point test by Moshkovitz and Raz [36] 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 28** (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 29** (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*

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

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

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

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 [36, Theorem 2] (with some observations from [36, Lemma 6.1]).

▶ **Lemma 30** (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*

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

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 [1] *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*

$$\Pr_{z, y_1, y_2} [\neg \mathit{SpacePoint}^{w,g}(z, y_1, y_2) \vee \exists Q \in \mathsf{near}^C_{\alpha'}(w) : Q|_{\mathit{span}\{z, y_1, y_2\}} \equiv g_{\mathit{span}\{z, y_1, y_2\}}] \geq 1 - \alpha$$

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

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 [36, Corollary 5.8], there is a partial converse to Lemma 30. Namely that these subspaces are a sampler.

▶ **Lemma 31** (Subfields Sample). *Let $\mathsf{dim}, \mathbb{F}, \mathbb{H}$, and deg be as in Lemma 30. 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 :*

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

5.2 Local Testers and Correctors

Now we can prove that the subspaces we discussed 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, $\mathsf{near}^C_\gamma(w)$ is the set of codewords that are in w for γ fraction of coordinates.

[1] This result can be found by looking into the proof of [36, Lemma 6.1] and observing that its f is $f(\gamma) = \gamma - 3\eta$ and setting d' to deg.

▶ **Lemma 32** (Pseudorandom Local Correctors). *Fix dimension $\mathit{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$, an integer ℓ and a degree deg. Denote $\eta = 2^7 \mathit{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{\mathit{dim}\cdot\mathit{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$.*

Then for any $\epsilon > 0$, relative agreement γ, and $c > 1$ such that $\gamma > \sqrt{\mathit{deg}\ell c/|\mathbb{F}|}$, 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 $|\mathbb{F}|^3 \mathit{poly}\left(\log(|\mathbb{F}|)\frac{|\mathbb{F}|}{\mathit{deg}}\right)$. There are $|\mathcal{I}| \leq |\mathbb{H}|^{2\mathit{dim}}$ functions. Further, these functions $\mathcal{I}$ are such that for any ℓ-string w we have:

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

$$\Pr_{f\in\mathcal{I}, i\in\mathbb{F}^{\mathit{dim}}}[y_i \notin f(w)_i] \leq \frac{1}{\epsilon^2}\left(\frac{1}{|\mathbb{H}|} + \frac{2}{|\mathbb{F}|}\right).$$

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

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

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 there is an extra $\frac{1}{|\mathbb{F}|}$ in the completeness error. The proof of completeness and soundness are in the full version. ◀

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 codewords list recovery algorithm from Lemma 27.

▶ **Corollary 33** (There Exists a Good Local Corrector). *Fix dimension $\mathit{dim} \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$ and a degree deg. Denote $\eta = 2^7 \mathit{dim}\left(\left(\frac{1}{|\mathbb{H}|}\right)^{1/8} + \left(\frac{\mathit{dim}\cdot\mathit{deg}}{|\mathbb{F}|}\right)^{1/4}\right)$. 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*

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

for $\mathcal{I}$ the family of functions from Lemma 32 there is a local corrector $f \in \mathcal{I}$ such that
Completeness:

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

Soundness:

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

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 Time-Space Efficient Deterministic List Decoding For Reed-Muller Codes

Now we can finally construct our list corrector for Reed-Muller codes. First, in Theorem 34 we will state our result for some conditions that are convenient to prove the result. Then in Theorem 35 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 34** (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 $f_i(w) = Q$.

Furthermore, the algorithm D makes $n^{1+4/a}L|\mathbb{H}|^{2a\mathsf{dim}}|\mathbb{F}|^{O(a^2)}\mathsf{poly}(\mathsf{dim})$ queries, runs 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 $\mathsf{poly}(|\mathbb{F}^a|\mathsf{dim}L)$.

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

▶ **Theorem 35** (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 $\gamma(n) \geq \frac{5\cdot 10^4}{\tau(n)n^{\tau(n)^3/c_0}}$ where c_0 is the large constant from Lemma 27. 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 $n^{O(\tau)}$.*

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 36** (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}.*

7 Open Problems

This is the first work showing that 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 [10] 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 [11] 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. The main bottleneck is the randomness-efficient low error local test of [36], which 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.
4. Are there time and space efficient deterministic list decoders for all local list correctable codes? In the unique decoding case [11] there are decoders for all locally correctable codes. Is it possible in the list decoding case?
5. Construct simultaneously time and space efficient pseudorandom generators. List decoding was used in prior PRGs [47, 42], and recent work has analyzed both the fine grained space complexity of PRGs [16, 15] and the fine grained time complexity of PRGs [8, 9, 14]. It is natural to attempt to get derandomization that is both very space efficient and very time efficient simultaneously.

References

1 M. Alekhnovich. Linear diophantine equations over polynomials and soft decoding of reed-solomon codes. *IEEE Transactions on Information Theory*, 51(7):2257–2265, 2005. doi:10.1109/TIT.2005.850097.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. doi:10.1145/278298.278306.

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

4 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, pages 485–495. Association for Computing Machinery, 1997. doi:10.1145/258533.258642.

5 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, pages 21–32. Association for Computing Machinery, 1991. doi:10.1145/103418.103428.

6 L.M.J. Bazzi and S.K. Mitter. Endcoding complexity versus minimum distance. *IEEE Transactions on Information Theory*, 51(6):2103–2112, 2005. doi:10.1109/TIT.2005.847727.

7 Louay Bazzi, Mohammad Mahdian, and Daniel A. Spielman. The minimum distance of turbo-like codes. *IEEE Transactions on Information Theory*, 55(1):6–15, 2009. doi:10.1109/TIT.2008.2008114.

8 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, pages 283–291. Association for Computing Machinery, 2021. doi:10.1145/3406325.3451059.

9 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)*, pages 125–136, 2022. doi:10.1109/FOCS52979.2021.00021.

10 Joshua Cook and Dana Moshkovitz. Explicit time and space efficient encoders exist only with random access. *The Proceedings of the 39th Computational Complexity Conference (CCC24)*, 2024. URL: https://eccc.weizmann.ac.il/report/2024/032/.

11 Joshua Cook and Dana Moshkovitz. Time and space efficient deterministic decoders. *STOC'25*, 2025. URL: https://eccc.weizmann.ac.il/report/2024/110/.

12 Nicolas T. Courtois. Efficient zero-knowledge authentication based on a linear algebra problem minrank. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 402–421, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-45682-1_24.

13 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*, pages 472–481, 2009. doi:10.1109/FOCS.2009.8.

14 Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *J. ACM*, 69(6), 2022. doi:10.1145/3555307.

15 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, pages 2039–2049. Association for Computing Machinery, 2024. doi:10.1145/3618260.3649772.

16 Dean Doron and Roei Tell. Derandomization with Minimal Memory Footprint. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference (CCC 2023)*, volume 264 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2023.11.

17 Peter Elias. List decoding for noisy channels. *Technical report (Massachusetts Institute of Technology. Research Laboratory of Electronics), No 335*, 1957.

18 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*, pages 2–12. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185341.

19 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)*, pages 57–, 1998. doi:10.1109/ISIT.1998.708638.

20 Oded Goldreich. *A Primer on Pseudorandom Generators*. University lecture series. American Mathematical Soc., 2010. URL: https://www.wisdom.weizmann.ac.il/~oded/prg-primer.html.

21 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. doi:10.1145/2699436.

22 André Gronemeier. A note on the decoding complexity of error-correcting codes. *Inf. Process. Lett.*, 100(3):116–119, 2006. doi:10.1016/j.ipl.2006.06.006.

23 Ofer Grossman and Dana Moshkovitz. Amplification and derandomization without slowdown. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 770–779, 2016. doi:10.1109/FOCS.2016.87.

24 V. Guruswami and M. Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999. doi:10.1109/18.782097.

25 Venkatesan Guruswami. Algorithmic results in list decoding. *Found. Trends Theor. Comput. Sci.*, 2(2):107–195, January 2007. doi:10.1561/0400000007.

26 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, pages 126–135. Association for Computing Machinery, 2003. doi:10.1145/780542.780562.

27 R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. doi:10.1002/j.1538-7305.1950.tb00463.x.

28 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

29 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, pages 131–140. Association for Computing Machinery, 2009. doi:10.1145/1536414.1536435.

30 Shankar Kalyanaraman and Christopher Umans. On obtaining pseudorandomness from error-correcting codes. In *FSTTCS'06*, pages 105–116. Springer-Verlag, 2006. doi:10.1007/11944836_12.

31 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, pages 80–86. Association for Computing Machinery, 2000. doi:10.1145/335305.335315.

32 R. Koetter and A. Vardy. Algebraic soft-decision decoding of reed-solomon codes. *IEEE Transactions on Information Theory*, 49(11):2809–2825, 2003. doi:10.1109/TIT.2003.819332.

33 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*, pages 2–10 vol.1, 1990. doi:10.1109/FSCS.1990.89518.

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

35 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, pages 15–23. Association for Computing Machinery, 2024. doi:10.1145/3618260.3649606.

36 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, pages 21–30. Association for Computing Machinery, 2006. doi:10.1145/1132516.1132520.

37 Dana Moshkovitz and Ran Raz. Two-query pcp with subconstant error. *J. ACM*, 57(5), June 2008. doi:10.1145/1754399.1754402.

38 Raphael Overbeck and Nicolas Sendrier. *Code-based cryptography*, pages 95–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-540-88702-7_4.

39 I.S. Reed. A brief history of the development of error correcting codes. *Computers & Mathematics with Applications*, 39(11):89–93, 2000. doi:10.1016/S0898-1221(00)00112-7.

40 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, pages 49–62. Association for Computing Machinery, 2016. doi:10.1145/2897518.2897652.

41 Nandakishore Santhi and Alexander Vardy. Minimum distance of codes and their branching program complexity. In *2006 IEEE International Symposium on Information Theory*, pages 1490–1494, 2006. doi:10.1109/ISIT.2006.262116.

42 Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, March 2005. doi:10.1145/1059513.1059516.

43 Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. doi:10.1145/359168.359176.

44 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. doi:10.1109/18.556667.

45 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, pages 388–397. Association for Computing Machinery, 1995. doi:10.1145/225058.225165.

46 Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, March 1997. doi:10.1006/jcom.1997.0439.

47 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001. doi:10.1006/jcss.2000.1730.

48 Luca Trevisan. Some applications of coding theory in computational complexity, 2004. URL: https://eccc.weizmann.ac.il/report/2004/043/.

49 Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003. doi:10.1016/S0022-0000(03)00046-1.

50 J.M. Wozencraft. List decoding. *Quterly Progress Report, Research Laboratory of Electronics*, 48:90–95, 1958.

51 Sergey Yekhanin. *Locally Decodable Codes*. Now Publishers Inc., Hanover, MA, USA, 2012.